TOWARDS GENETIC ALGORITHM METHODOLOGY IN CHEMOMETRICS

Carlos B. Lucasius, Jr.

TOWARDS
GENETIC ALGORITHM METHODOLOGY
IN
CHEMOMETRICS

Manuscript commission : Dr. L. Davis and Dr. W.J. Melssen

TOWARDS
GENETIC ALGORITHM METHODOLOGY
IN
CHEMOMETRICS

een wetenschappelijke proeve op het gebied van de natuurwetenschappen

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op
woensdag 6 oktober 1993
des namiddags te 1.30 uur precies

door

Carlos Borromeo Lucasius

geboren op 23 mei 1959
te Curaçao, Nederlandse Antillen

FEBO druk – Enschede

Promotor	Prof. Drs. G. Kateman
Co-promotor	Dr. L.M.C. Buydens

The investigations reported in this thesis were carried out at the Laboratory for Analytical Chemistry, Faculty of Science, at the Katholieke Universiteit Nijmegen under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007.

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Lucasius, Carlos Borromeo (Jr.)

Towards genetic algorithm methodology in chemometrics /
Carlos Borromeo Lucasius, Jr. – [S.l. : s.n.] (Enschede :
FEBO druk). – Ill.
Proefschrift Nijmegen. – Met lit. opg. – Met samenvatting
in het Nederlands.
ISBN 90-9006312-9
Trefw.: genetische algoritmen / chemometrie / analytische
chemie.

Contents

Dankbetuiging / Acknowledgment / Gradisimentu

Op deze plaats wil ik iedereen bedanken die in wetenschappelijk en/of moreel opzicht heeft bijgedragen aan de totstandkoming van dit proefschrift. Speciale dank gaat uit naar mijn ouders, mijn zussen, en mijn geliefde.

At this place, I would like to thank everyone who has contributed in a scientific and/or moral sense to the completion of this thesis. Special thanks goes to my parents, my sisters, and my beloved.

Na e luga aki mi ke gradisi tur hende ku a kontribuí den un sentido sientífiko i/o moral pa realisashon di e tesis aki. Mi gratitut ta bai speshalmente na mi mayornan, mi ruman muhénan, i mi amor.

Carlos

CHAPTER I

Chemometrics

Contents

Chemometrics

This chapter presents the historical development of chemometrics in a nutshell, with the primary aim to meet the group of genetic algorithm scientists (and other readers) who are unacquainted with this discipline.

Introduction

Analytical chemistry is concerned with the extraction of relevant information from objects – systems or processes – that have a chemical nature or are of chemical importance. Two main stages can be distinguished in analytical-chemical procedures. In the first stage, data are *collected* through chemical-, physical-, or physico-chemical measurements performed on the object, using dedicated instruments; in some cases, the object undergoes a special pre-preparation in order to make it more suitable for performing the measurements. In the second stage, the data are *processed* (interpreted) in order to make explicit the relevant implicit information.

During the past two decades, growing economical and social demands for more and faster available chemical information have lead to increasing instrumental complexity, e.g. multivariate measurement. In the wake of these developments, numerous sophisticated data processing techniques have emerged. This collection, or arsenal, of computational methods has established itself as a chemical discipline which became generally known as *chemometrics* [7, 10]. Loosely, chemometrics may be regarded as the "dry" (non-instrumental) part of analytical chemistry, aimed at optimal processing of chemical data.

Contrary to what the preceding may suggest, however, it is important to realize that the computational methods comprising chemometrics are certainly not confined to analytical-chemical problem solving. Indeed, numerous computational methods that have been qualified as "chemometric" in the chemical literature, are also encountered in disciplines such as biometrics, econometrics, sociometrics, psychometrics, etc. The "-metrics" disciplines have in common that they traditionally borrow substantially, according to prevailing demands, from more general computational disciplines, e.g. experimental statistics.

Up to now, the evolution of chemometrics [5] has featured the following three main stages, incrementally:

1. Numerical computation;
2. Symbolic computation;
3. Natural computation.

1 Numerical computation

Chemometrics nucleated as a collection of mostly numerical computation methods, *viz.* calculation methods based on mathematical, often statistical, models. New numerical computation methods were incessantly introduced ever since, contributing substantially to the fast growth of chemometrics up to now. An excellent standard treatise on chemometrics as an arsenal of mostly numerical computation methods can be found in [7].

A potential drawback of numerical computation methods is that they incorporate strong assumptions – e.g. linearity and continuity – for the sake of mathematical tractability. As long as these assumptions are in reasonable keeping with the analytical-chemical system or process under consideration, reliable results are generally to be expected. However, many real-world cases exist wherein the assumptions are invalid, so that the reliability of results comes into question.

2 Symbolic computation

Symbolic computation methods – collectively comprising the discipline *artificial intelligence* [6] – were introduced into chemometrics shortly after their inception in the late 1970ties. These methods were welcomed for their promise to solve discontinuous problems that had been pressing for some time already, as available numerical methods turned out to be inadequate due to their continuous nature.

Artificial intelligence methods mimick the heuristic reasoning and decision making of a human expert in the field of interest; accordingly,

specific implementations of these methods have been called expert systems. The information processing in an expert system proceeds along chains of logical rules derived from the expert's domain knowledge made explicit. These rules involve discontinuous (i.e. qualitative) entities, or symbols, e.g. "sample", "instrument", "procedure", etc. – hence symbolic computation.

An advantage of expert systems lies in automating the heuristic reasoning and decision making of the expert, i.e. in making the specialized knowledge of a single individual available to a broader public. A disadvantage, however, lies in the fact that any flaws, biases, and voids in this knowledge are inherited by the system – errors which reveal the subjective nature of human knowledge. In addition, the extraction of human knowledge turns out to be a wearisome and highly error-prone procedure. Attempts to resolve these difficulties (e.g. adaptive expert systems) have met only meager success, in general.

3 Natural computation

Natural computation methods – collectively comprising the discipline *natural intelligence* (as it is informally termed to date) – were introduced into chemometrics in the late 1980ties. These methods are, as their name indeed suggests, inspired by a natural system or process. For instance, genetic algorithms – the subject of this thesis – are adaptive optimization methods which simulate a biological evolution process (see Chapter II). Other well-established natural computation methods are artificial neural networks [8, 9] (adaptive learning systems which mimick the brain) and simulated annealing [1, 8] (adaptive optimization which simulates the gradual cooling of a solid to overcome local minima in energy caused by strain and crystal imperfections). The former were introduced into chemometrics by, inter alia, [2, 3, 11, 12], and the latter by, inter alia, [4].

In contrast with numerical- and symbolic computation methods, which traditionally employ a deterministic problem solving strategy based on strong domain assumptions (i.e. on supposedly true knowledge), natural computation methods employ a probabilistic problem solving strategy based on weak or moderate assumptions. More precisely, natural computation methods distinguish themselves in that they perform steps in the problem space that are controlled by comparatively simple, plausible, probabilistic transition rules. Free of strong domain assumptions, natural computation methods can produce reliable results, even when the target problem is too complex for a numerical or symbolic approach. This advantage comes with a price tag, though: the computational expenses are usually comparatively high. Not surprisingly, therefore, the recent upsurge in the application of natural computation methods is to be ascribed in great part to developments towards cheaper and faster (e.g. parallel) computing systems during the past few years. As these developments seem to continue at a high pace, larger and larger applications quickly become practically feasible.

The success of natural computation is generally attributed to the non-linear way in which the target problem is solved. In many cases, such non-linearities arise naturally from interactions between comparatively simple memory units that comprise the medium in which the information accumulated during the problem solving process is stored; these interactions are governed by aforementioned probabilistic transition rules. (Some more detail on natural computation in general is provided in Chapter 1.)

Conclusions

Chemometrics comprise an arsenal of computational methods for the processing of chemical data. These methods are borrowed mainly from other disciplines, according to prevailing demands; in addition, by combining existing methods (or parts thereof) in a meaningful way, new useful computational methods may emerge. Recently, chemometrics has borrowed methods from the promising new realm of natural computation, among which genetic algorithms.

References

[1] Aarts, E.H.L. and Korst, J.H.M. *Simulated Annealing and Boltzmann Machines. A Stochastic Approach to Combinatorial Optimization and Neural Computing.* Wiley, Chichester, 1989.

[2] Bos, A. *Artificial Neural Networks as a Tool in Chemometrics.* PhD thesis, Technische Universiteit Twente, Enschede, the Netherlands, 1993.

[3] Bos, M., Bos, A., and Linden van der, W.E. Processing of ion-selective electrode array signals by a neural network. *Analytica Chimica Acta*, 233:31–39, 1990.

[4] Kalivas, J H. Optimization using variations of simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 15:1–12, 1992.

[5] Kateman, G. Evolutions in chemometrics. *The Analyst*, 115:487–493, 1990.

[6] Luger, G.F. and Stubblefield, W.A *Artificial Intelligence and the Design of Expert Systems.* The Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1989

[7] Massart, D L., Vandeginste, B.G.M., Deming, S.N , Michotte, Y., and Kaufman, L. *Chemometrics: A textbook.* Elsevier, Amsterdam, 1988.

[8] Reeves, C.R., editor. *Modern Heuristic Techniques for Combinatorial Problems.* Advanced Topics in Computer Science. Blackwell Scientific Publications, Oxford, 1993.

[9] Rumelhart, D.E., Feldman, J.A, and Hayes, P.J., editors. *Parallel Distributed Processing.* MIT Press, Cambridge, MA, 1986.

[10] Sharaf, M A., Illman, D.L., and Kowalski, B.R. *Chemometrics*, volume 82 of *Chemical Analysis.* Wiley, New York, NY, 1986.

[11] Smits, J.R.M. *Exploring the Possibilities of Applying Artificial Neural Networks on Problems in Analytical Chemistry.* PhD thesis, Katholieke Universiteit Nijmegen, Nijmegen, the Netherlands, 1993.

[12] Zupan, J. and Gasteiger, J. Neural networks: A new method for solving chemical problems or just a passing phase? *Analytica Chimica Acta*, 248.1-30. 1991.

CHAPTER II

Genetic algorithm methodology

Contents

Genetic algorithm methodology

This chapter outlines the background, principles, and scope of genetic algorithms. There is mounting empirical evidence that genetic algorithms are competitive with regard to tackling complex, large-scale optimization problems. However, the methodology is comparatively young and as yet not well accessible.

1 Background

The relative infancy of genetic algorithms is immediately apparent from the scientific literature on the subject: the first international conference on genetic algorithms was held only in 1985 [5]; the first comprehensive textbook on the subject was published only in 1989 [4]; and a journal dedicated to this particular scientific field did not exist until the completion of the research described in this thesis.

The relatively poor status of the literature on genetic algorithms is among the main reasons why, for nearly two decades, wider adherence of genetic algorithm methodology in the applied sciences has remained inhibited – especially outside the USA, the country where the genetic algorithm concept originated and still appears to enjoy wider adherence in different scientific areas. In addition, since genetic algorithm methodology originates from fundamental computing sciences rather than from applied sciences, the mainstream literature traditionally lays considerably more emphasis on fundamental issues than on the practicalities of application – apparently posing a "cultural gap" towards potential users in the applied sciences.

Nonetheless, it turns out that developments in the science of genetic algorithms have recently exceeded some critical point, as they are presently gaining grounds dramatically, worldwide. Other strong indicators of the wider recognition of genetic algorithms science exist, related to these developments. For instance, the progenitor of genetic algorithms – John H. Holland – has received one of 1992's MacArthur Genius Awards. (This prestigious type of fellowship was also awarded to D.E. Rumelhart for his work on artificial neural networks a few years ago.) Furthermore, the first issue of the journal *Evolutionary Computation* has appeared in 1993; this journal may be considered the first major step towards an international forum for facilitating and enhancing the exchange of information among researchers involved in both the theoretical and practical aspects of computational systems of an evolutionary nature.

An impression of the interest in genetic algorithms present in non-USA countries, can be inferred from the following statistics on membership of the established international electronic mailing list GA-LIST@AIC.NRL.NAVY.MIL (source *ibid.*; January, 1993):

COUNTRY NAME	# MEMBERS
Argentina	1
Australia	59
Austria	10
Belgium	9
Canada	64
Czechoslovakia'	1
Denmark	14
Finland	8
France	36
Germany	65
Greece	1
Hungary	2
Ireland	6
Israel	6
Italy	21
Japan	38
Korea	10
Mexico	13
Netherlands	32
New Zealand	7
Portugal	2
Spain	15
Sweden	16
Taiwan	8
United Kingdom	148
USSR'	2
Yugoslavia'	1

(A prime denotes "former".)

The recent worldwide upsurge in the science of genetic algorithms is to be ascribed in great part to the growing body of reported successes of an unprecedented nature obtained by scientists

in distinct fields of real-world application (including computational chemistry) after embarking on the interdisciplinary challenge. In view of these promising developments, it is reasonable to expect that genetic algorithm methodology is going establish itself for a considerable time span.

2 Principles

Genetic algorithms are among so-called *simulated evolution* methods problem solving methods that simulate a natural evolution process. The mimicked evolution process comprises generalized evolutionary components such as: an ensemble, or population, comprised of artificial creatures; competition between these creatures on the basis of their observed quality, or fitness; reproduction and modification of the fittest among all creatures to create new creatures who are expectedly fitter, on average, and replace the creatures in the population who have reached the end of their life-time.

The artificial creatures in such a simulated evolution process are computer-coded candidate solutions to a large, complex problem. Without prior knowledge about the problem, these candidate solutions are initially arbitrary, as a rule. They are iteratively updated by the constituent procedures – among which one that acts as an arbiter who judges the relative quality of the candidate solutions, and as such plays the role of an artificial environment wherein some may survive whilst others will not. Altogether, the interplay between the evolutionary components should cause gradual improvement, which may ultimately culminate in a solution to the problem.

The intuitive appeal of natural evolution as a self-contained improvement process derives from the simplicity of the underlying rules – postulated by Darwin [1] succinctly as "struggle for life" (competition rule) and "survival of the fittest" (selection rule); these rules are concerned with the *exploitation* of useful information contained in the population of individuals concerned. In addition, there are rules which prescribe how new creatures may arise from existing creatures (modification rules), e.g. through spontaneous mutation or through sexual reproduction; these rules are concerned with the *exploration* of the space of possibilities.

Importantly, the interplay between exploration and exploitation is of a synergic nature, from which all evolution processes – natural and simulated – derive their search power. As for natural evolution, such an interplay was persuasively voiced by the English scientist Richard Dawkins [3] through the following philosophical observations:

> *Living things are too improbable and too beautifully 'designed' to have come into existence by chance. How, then, did they come into existence? The answer, Darwin's answer, is by gradual, step-by-step transformation from simple beginnings, from primordial entities sufficiently simple to have come into existence by chance. Each successive change in the gradual evolutionary process was simple enough, relative to its predecessor, to have arisen by chance. But the whole sequence of cumulative steps constitutes anything but a chance process, when you consider the complexity of the final end-product relative to the original starting point. The cumulative process is directed by non-random survival.*

That the simple rules of the evolutionary game are apparently capable of orchestrating a process that accomplishes amazingly complex natural design, as testified by all lively precedence on earth, has, quite understandably, been a source of inspiration for many scientists – motivating them to simulate the process on the computer in order to accomplish complex computation. Among these scientists, one encounters not only abovementioned Holland, who gave birth to genetic algorithm methodology in the early sixties at the University of Michigan. Other scientists have around the same time pioneered different simulated evolution methodologies (mentioned in Chaper III). However, to draw a discriminating line between the simulated evolution methodologies seems dubious (if not meaningless), since there are so many common features.

3 Application scope

Genetic algorithms have indeed been invented primarily to tackle large, complex problems – not to model evolution itself. Thereby, merely those principles from natural evolution are borrowed that are considered useful or convenient for some

good reason, e.g. for search efficiency or for ease of implementation. Moreover, the borrowed principles are not implemented in a dogmatic way, i.e. they are not considered as fixed tools within which limits an application must be developed, but instead are regarded as environments that can be adjusted to match the task at hand. So, like other natural computation methods, genetic algorithms are examples of the plausible assumption that what nature does is not necessarily exactly the best thing to do when it comes to problem solving. This said, evolutionary principles should, however, still be regarded as the main driving force behind the search power of genetic algorithms.

Based only loosely on evolutionary principles, genetic algorithms fulfil a very general algorithmic description, i.e. a description which does not pinpoint to a single algorithm or a few modifications thereof, but rather spans a large family of algorithms – genetic algorithms, by definition. Given the empirically established fact that this family competitively covers a wide range of problems, genetic algorithm methodology is widely praised as versatile, i.e. as having a wide application scope. Equivalently, genetic algorithms are widely touted for their adaptability: a genetic algorithm dedicated to a particular problem can be customized to approach an entirely different problem. On the other hand, the versatility of genetic algorithm methodology can be a drawback too. For, configuration – or, basically, the selection of the best genetic algorithm (from the family) for a particular problem of interest – is not straightforward, it is in fact wearisome, and normally demands a considerable experience of application.

4 Conclusions

Genetic algorithms comprise a large family of computational methods based upon principles of evolution according to Darwin. The promise of this family, or genetic algorithm methodology, resides in its competitiveness with regard to solving large, complex problems encountered in many distinct fields of real-world application. Their versatility is widely praised, but comes with a price: the task to find an optimal configuration is difficult, time-consuming, and demands a considerable experience of application.

In view of the relative infancy and extensive nature of genetic algorithm methodology, it is crucial for the novice practitioner to start with the acquisition of fundamental concepts of general practical importance. Only recently, literature with such focus has started to appear, e.g. [2, 6, 7, 8], and Parts 1 and 2 of this thesis.

References

[1] Darwin, C. *The Origin of Species by Means of Natural Selection: the Preservation of Favoured Races in the Struggle for Life.* John Murray, London, 1859. First edition text, edited with an introduction by J.W. Burrow, published in Pinguin Books 1968.

[2] Davis, L., editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, NY, 1991. Part I: A Genetic Algorithms Tutorial, p. 1–101.

[3] Dawkins, R. *The Blind Watchmaker.* Pinguin Books, London, 1986.

[4] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] Grefenstette, J.J., editor. *Proceedings of the First International Conference on Genetic Algorithms,* Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[6] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Artificial Intelligence Series. Springer-Verlag, Berlin, 1992.

[7] Reeves, C.R., editor. *Modern Heuristic Techniques for Combinatorial Problems.* Advanced Topics in Computer Science. Blackwell Scientific Publications, Oxford, 1993.

[8] Whitley, D. A genetic algorithm tutorial. Technical Report CS-93-103, Colorado State University, March 1993.

CHAPTER III

Objectives, scope, and strategy

Contents

Objectives, scope, and strategy

This chapter presents the objectives, scope, and strategy of the research described in this thesis, carried out in the period from June 1988 to July 1992.

1 Objectives

The first objective of the research described in this thesis is to acquire a thorough understanding of genetic algorithm methodology insofar as it can serve as a practically useful tool in chemometrics. The second objective of the research is to develop such a tool for use on computers. The third objective of the research is to validate this tool through application to analytical-chemical problems. Altogether, it is aimed that these endeavors should ultimately benefit the community of chemometricians and other computational chemists.

In view of the innovative nature of these objectives, combined with the fact that genetic algorithm methodology is extensive and presently still amidst the rapids of its practical maturation, most research activities have purposefully been targeted towards laying a broad foundation for future, in-depth application to suitable computational problems of various kinds within analytical chemistry.

1.1 A unifying conceptual framework for application

An indispensible part of abovementioned foundation should be a well-structured, unifying conceptual framework of general practical importance. The need for such a framework becomes apparent from, inter alia, the following observations made by Reeves [14]:

> To some extent the development of GA practice has been rather chaotic, and the terminology has not had time to settle down. It would also appear that in a number of cases similar ideas have been 'discovered' independently by more than one person.

The relative infancy in which genetic algorithm methodology finds itself to date with regard to the development of applications, may be ascribed in great part to the poor communication between practitioners and theoreticians: up to now, most theoreticians have shown relatively little interest in the practicalities of application, whereas many practitioners have often questioned the practical usefulness of certain mathematical models proposed by theoreticians. This regrettable situation is further aggravated by the poor communication between practitioners mutually: most practitioners have been predominantly concerned with their own, specific applications; ironically, this situation is to be ascribed in great part to the versatility of genetic algorithm methodology: practitioners engaged in different fields of application are generally not much inclined to engage in exchanging ideas.

Since recently, the situation is gradually changing for the better. In particular, a number of practitioners have taken up the difficult but worthwhile task to make explicit unifying concepts of general importance to the application of genetic algorithms. Examples of literature with that focus were given in the preceding chapter. This thesis too, aims at contributing in that respect (Parts 1 and 2), be it in its own distinctive way.

2 Scope

Genetic algorithm methodology in its entirety – its varied theoretical foundations and its ample application scope (versatility) – can not possibly be covered by research with a span of only four years.

For this reason, theoretical issues of the methodology have deliberately been left underexposed, in keeping with abovementioned objectives; undue mathematical rigor is avoided. Instead, emphasis lies on making explicit – in a relatively simple, not mainly formal, hopefully intuitively appealing way – the unifying concepts of general importance to the application of genetic algorithms; this serves abovementioned intention to improve the computational chemist's access to applied genetic algorithm methodology.

For the same reason, other evolution-based methodologies – e.g. evolution strategies [12, 13,

15], evolutionary programming [3, 4, 5, 6], genetic programming [1, 10, 11], and classifier systems [2. 7, 8, 9] – are not taken up for application Instead, these methodologies are only briefly passed in review (see Chapter 1), merely to put genetic algorithm methodology in a broader perspective

3 Strategy

The research strategy adhered to, is clearly reflected in the organization of this thesis in parts

Part 1 – ACQUISITION – represents the extraction of practically relevant aspects of the methodology from the mainstream literature, and their formulation into abovementioned unifying conceptual framework of know-how aimed at facilitating the application to real-world problems This "enrichment process" is motivated by the fact that the practically relevant knowledge is only scarcely present in the literature, and, in addition, very much distributed in bits and pieces in an often implicit form.

Part 2 – DEVELOPMENT – is predominantly concerned with the conversion of the knowledge, as it was acquired (i e made explicit and organized) in Part 1, into domain-independent algorithmic procedures for use on computers The discussion also elaborates on the practicalities of combining such procedures with domain-dependent algorithmic procedures in order to obtain the aimed genetic algorithm application in executable software

Part 3 – APPLICATION – represents the proof-of-principle validation of the methodology, as it was developed in Part 2, by way of its application to a number of representative real-world problems in analytical chemistry These projects were conducted mostly in collaboration with specialists in the field of applicication, in order to accomplish greater productivity through mutual complementarity.

4 Conclusions

In response to the shortage of application-oriented literature and software on genetic algorithms that has prevailed during the research described in this thesis Parts 1 and 2 describe the creation of a broad, practicable foundation of genetic algorithm methodology, aimed at future in-depth application The viability and competitiveness of this foundation with respect to analytical-chemical problem solving, is assessed in Part 3

References

[1] Angeline, P J and Pollack, J B Competitive environments evolve better solutions for complex tasks In Goldberg, D E and Schaffer, J D , editors, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993 Morgan Kaufmann Submitted

[2] Booker, L B , Goldberg, D E , and Holland, J H Classifier systems and genetic algorithms *Artificial Intelligence*, 40 235, 1989

[3] Fogel, D B An evolutionary approach to the traveling salesman problem *Biological Cybernetics*, 60 139, 1988

[4] Fogel, D B and Atmar, J W Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems *Biological Cybernetics*, 63 111 1990

[5] Fogel, D B , Fogel, L J , and Porto V W Evolving neural networks *Biological Cybernetics*, 63 487, 1990

[6] Fogel, L J , Owens, A J , and Walsh, M J *Artificial Intelligence through Simulated Evolution* Wiley, New York, NY, 1966

[7] Goldberg, D E and Holland, J H Guest editorial Genetic algorithms and machine learning *Machine Learning*, 3 95–99, 1988

[8] Holland, J H Escaping brittleness The possibilities of general-purpose learning algorithms applied to parallel rule-based systems In Caianello et al , editor *Machine Learning II*, page 593 Morgan Kaufmann, 1986

[9] Holland, J H , Holyoak, K J , Nisbett, R E , and Thagard, P R *Induction Processes of Inference, Learning and Discovery* Computational Models of Cognition and Perception MIT Press, Cambridge, MA, 1986

[10] Koza, J R Concept formation and decision tree induction using the genetic programming paradigm In Schwefel, H P and Manner, R , editors, *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, pages 124–128, Berlin, 1991 Springer-Verlag Lecture Notes in Computer Science 496

[11] Koza, J R *Genetic Programming On the Programming of Computers by Means of Natural Selection* MIT Press, Cambridge, MA, 1992

[12] Rechenberg, I. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* Frommann-Holzboog Verlag, Stuttgard-Bad, 1973.

[13] Rechenberg, I. The evolution strategy. A mathematical model of Darwinian evolution. In Frehland, E., editor, *Synergetics - From Microscopic to Macroscopic Order*, page 122, Berlin, 1984. Springer-Verlag.

[14] Reeves, C.R., editor. *Modern Heuristic Techniques for Combinatorial Problems.* Advanced Topics in Computer Science. Blackwell Scientific Publications, Oxford, 1993.

[15] Schwefel, H.-P Evolution strategies. A family of non-linear optimization techniques based on imitating some principles of organic evolution. *Annals of Operations Research*, 1:165, 1984.

CHAPTER IV

Selection from publications

Contents

Selection from publications

1 Selection

The body of this thesis comprises a reasonably representative selection from the publications listed in Section 2 below.

The selection was realized according to the following table:

Part I:

Chapter 1:	publication 5
Chapter 2:	publication 6

Part II:

Chapter 3:	publication 7
Chapter 4:	publication 8
Chapter 5:	publication 9

Part III:

Chapter 6:	publication 13
Chapter 7:	publication 14
Chapter 8:	publication 26
Chapter 9:	publication 24
Chapter 10:	publication 12
Chapter 11:	publication 22
Chapter 12:	publication 3
Chapter 13:	publication 1

Adaptations to the selected publications have been kept to a minimum, so that the chapters are to a considerable extent self-contained. As a consequence, there is some overlap between the chapters (e.g. general introductions to meet readers who lack the necessary background in genetic algorithms), and some inconsistency in lay-out, writing style, and the use of terminology.

2 Publications

The research activities performed by the author of this thesis have lead to the following publications (existing or to appear) in refereed scientific literature:

1. C.B. Lucasius, M.L.M. Beckers, G. Kateman, *Genetic Algorithms in Wavelength Selection: A Comparative Study.* ANALYTICA CHIMICA ACTA, 1993 (to appear)

2. C.B. Lucasius, M.J.J. Blommers, L.M.C. Buydens and G. Kateman, *A Genetic Algorithm for Conformational Analysis of DNA.* In L. Davis (Ed.), THE HANDBOOK OF GENETIC ALGORITHMS, Chapter 18:251–281, 1991. Van Nostrand Reinhold

3. C.B. Lucasius, A.D. Dane and G. Kateman, *On k-Medoid Clustering Of Large Data Sets with the Aid of a Genetic Algorithm: Background, Feasibility and Comparison.* ANALYTICA CHIMICA ACTA, 1993 (to appear)

4. C.B. Lucasius, A.H.C. van Kampen, L.M.C. Buydens and G. Kateman, *On the Robustness of Genetic Algorithms in Noisy Optimization: An Application in Non-Linear Compaction of Variates.* CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS, 1993 (to appear)

5. C.B. Lucasius and G. Kateman, *Understanding and Using Genetic Algorithms. Part 1. Concepts, Properties and Context.* CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS 19:1–33, 1993

6. C.B. Lucasius and G. Kateman, *Understanding and Using Genetic Algorithms. Part 2: Representation, Configuration and Hybridization.* CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS, 1993 (to appear)

7. C.B. Lucasius and G. Kateman, *GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 1. General Perspective.* COMPUTERS & CHEMISTRY, 1993 (to appear)

8. C.B. Lucasius and G. Kateman, *GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 2: Toolbox Description.* COMPUTERS & CHEMISTRY, 1993 (to appear)

9. C.B. Lucasius and G. Kateman, *Towards Solving Subset Selection Problems with the*

Aid of the Genetic Algorithm. In R. Männer and B. Manderick (Eds.), PARALLEL PROBLEM SOLVING FROM NATURE 2:239–247, 1992. Elsevier (Brussels, September 28–30 — European Commission DG XIII, ESPRIT Basic Research; National Fund for Scientific Research; Parsytec GmbH; Research Council of the Free University Brussels; Siemens-Nixdorf Belgium)

10. C.B. Lucasius and G. Kateman, *Application of Genetic Algorithms in Chemometrics.* In J.D. Schaffer (Ed.), PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS AND THEIR APPLICATIONS 170–176, 1989. Morgan Kaufmann (Fairfax, June 4–7, 1989 — Navy Center for Applied Research in Artificial Intelligence; Naval Research Laboratory; Philips Laboratories, North American Philips Corporation)

11. C.B. Lucasius and G. Kateman, *Genetic Algorithms for Large-Scale Optimization in Chemometrics: An Application.* TRENDS IN ANALYTICAL CHEMISTRY 10(8):254–261, 1991

12. C.B. Lucasius, A.P. de Weijer, L.M.C. Buydens and G. Kateman, *CFIT: A Genetic Algorithm for Survival of the Fitting (software description).* CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS 19:337–341, 1993

13. C.B. Lucasius, S. Werten, A.H.J.M. van Aert, M.J.J. Blommers and G. Kateman, In H.-P. Schwefel and R. Männer (Eds.), *Conformational Analysis of DNA using Genetic Algorithms.* PARALLEL PROBLEM SOLVING FROM NATURE 1:90–97, 1991. Springer-Verlag. Lecture Notes in Computer Science 496 (Dortmund, October 1–3, 1990 — Parsytec GmbH; Paracom GmbH; Society of the Friends of the University of Dortmund e.V.; IBM Deutschland GmbH; Siemens AG; The Computer Society of IEEE; Ministry of Science and Research of Nordrhein-Westfalen)

14. M.J.J. Blommers, C.B. Lucasius, G. Kateman and R. Kaptein, *Conformational Analysis of a Dinucleotide Photodimer with the Aid of the Genetic Algorithm.* BIOPOLYMERS 32:45–52. 1992

15. P. van Bommel, C.B. Lucasius and Th.P. van der Weide *Genetic Algorithms for Optimal Database Design.* INFORMATION AND SOFTWARE TECHNOLOGY, 1993 (to appear)

16. D.C. van Leijenhorst, C.B. Lucasius, J.M. Thijssen, *Genetic Algorithms in Optical Design.* EVOLUTIONARY COMPUTATION, 1993 (to appear)

17. D.C. van Leijenhorst, C.B. Lucasius, J.M. Thijssen, *Genetic Algorithms in Optical Design.* In R.A. de Groot and J. Nadrchal (Eds.), PHYSICS COMPUTING, PROCEEDINGS OF THE 4^{th} INTERNATIONAL CONFERENCE 389–390, 1993. World Scientific (Prague, August 24–28, 1992 — Computational Physics Group of the European Physical Society; German Physical Society; Cray Research; Siemens-Nixdorf; Academic Press; Parsytec; Institute of Physics of the Czechoslovac Academy of Sciences; Faculty of Mathematics and Physics of the Charles University; Faculty of Nuclear Science and Physical Engineering of the Czech Technical University, Praha; Physical Scientific Section of the Union of the Czech Mathematicians and Physicists)

18. T.-H. Li, C.B. Lucasius and G. Kateman, *Optimization of Calibration Data with the Dynamic Genetic Algorithm.* ANALYTICA CHIMICA ACTA 268(1):123–134, 1992

19. R.M. Lopes Marques, P.J. Schoenmakers, C.B. Lucasius and G. Kateman, *Modelling Chromatographic Behaviour as a Function of pH and Solvent Composition in RPLC.* CHROMATOGRAPHIA 36:83–95, 1993

20. A. Parczewski, C.B. Lucasius and G. Kateman, *Evolutionary Determination of Physico-Chemical Parameters and Concentrations of Analytes From Titration Data.* FRESENIUS JOURNAL OF ANALYTICAL CHEMISTRY, 1993 (to appear)

21. R. Wehrens, C.B. Lucasius, L.M.C. Buydens and G. Kateman, *HIPS, a Hybrid Self-Adapting Expert System for NMR Spectrum Interpretation Using Genetic Algorithms.* ANALYTICA CHIMICA ACTA 277:313–324, 1993

22. R. Wehrens, C.B. Lucasius, L.M.C. Buydens and G. Kateman, *Sequential Assignment of 2D-NMR Spectra of Proteins Using Genetic Algorithms.* JOURNAL OF CHEMICAL INFORMATION AND COMPUTER SCIENCES 33(2):245–251, 1993

23. A.P. de Weijer, C.B. Lucasius, L.M.C. Buydens, G. Kateman and H.M. Heuvel, *Using Genetic Algorithms for an Artificial Neural Network Inversion.* CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS, 1993 (to appear)

24. A.P. de Weijer, C.B. Lucasius, L.M.C. Buydens, G. Kateman, H.M. Heuvel and H. Mannee *A New Approach to Curve-Fitting Using Natural Computation* ANALYTICAL CHEMISTRY, 1993 (to appear)

25. D. Wienke, C.B. Lucasius and G. Kateman, *Multicriteria Target Vector Optimization of Analytical Procedures Using a Genetic Algorithm. Part 1. Theory, Numerical Simulations and Application to Atomic Emission Spectroscopy.* ANALYTICA CHIMICA ACTA 265(2):211–225, 1992

26. D. Wienke, C.B. Lucasius, M. Ehrlich and G. Kateman, *Multicriteria Target Vector Optimization of Analytical Procedures Using a Genetic Algorithm. Part 2. Polyoptimization of the Photometric Calibration Graph of Dry Glucose Sensors for Quantitative Clinical Analysis.* ANALYTICA CHIMICA ACTA 271:253–268, 1993

PART I

ACQUISITION

CHAPTER 1

Understanding and Using Genetic Algorithms. Part 1. Concepts, Properties and Context

Contents

Chemometrics and Intelligent Laboratory Systems, 19 (1993) 1–33
Elsevier Science Publishers B V., Amsterdam

Understanding and using genetic algorithms

Part 1. Concepts, properties and context

C.B. Lucasius and G. Kateman

Laboratory for Analytical Chemistry, Faculty of Science, Katholieke Universiteit Nijmegen (Netherlands)

(Received 8 June 1992; accepted 20 July 1992)

Abstract

Lucasius, C B. and Kateman, G., 1993 Understanding and using genetic algorithms Part 1 Concepts, properties and contexts. *Chemometrics and Intelligent Laboratory Systems*, 19. 1–33.

Genetic algorithms are search algorithms founded upon the principles of natural evolution laid down by Darwin They turn out to be competitive for a certain class of problems – complex large-scale problems, as a rule Among the favorable properties of genetic algorithms are efficiency, robustness and versatility. A less favorable property of genetic algorithms is the imprecision as a result of the noise used by the method This tutorial consists of two parts which treat a variety of key issues concerning genetic algorithms The first part emphasizes the principles underlying genetic algorithms, their search characteristics and the broader perspective in which they fit This serves as a general, comprehensive introduction Starting from the first part, the second part of this tutorial elaborates on practical issues such as representation, configuration and hybridization with other techniques Thereby, some hands-on information is provided, so that common pitfalls can be avoided in using a methodology that exhibits its full power only when handled according to the principles it is based upon

CONTENTS

Correspondence to Dr. C B. Lucasius, Laboratory for Analytical Chemistry, Faculty of Science, Katholieke Universiteit Nijmegen (Netherlands).

PREFACE

Genetic algorithms turn out to be highly suited to tackling complex, large-scale optimization problems in an efficient and efficacious way. The science is exciting and promising, but still comparatively young. The first international conference on genetic algorithms was held only in 1985 [1], the first comprehensive textbook on the subject was published only in 1989 [2], and a journal dedicated to this particular scientific field does not yet exist. Perhaps not surprisingly, therefore, access to the literature on genetic algorithms is still relatively sparse and diffuse. In addition, genetic algorithm methodology stems from the fundamental computing sciences rather than from the applied sciences. In the mainstream literature this has thus far led to a strong emphasis on fundamental issues, apparently imposing a 'cultural gap' towards the applied sciences. Altogether, these circumstances are among the main causes that have for nearly two decades obstructed wider adherence in applied sciences — much more so outside the United States of Amer-

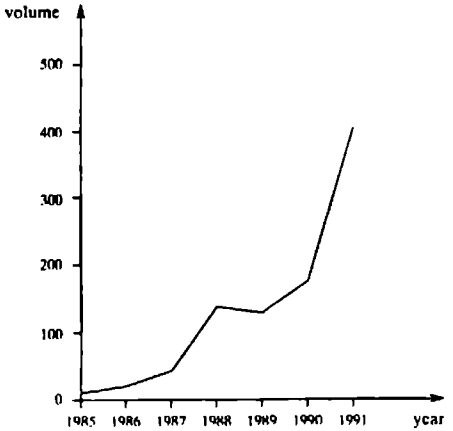

Fig. 1. Volume of recorded publication entries on genetic algorithms over the period 1985–1991. Source: the official literature data bases COMPENDIX•PLUS, NTIS, INSPEC-2, HARD SCIENCES, ARTIFICIAL INTELLIGENCE, MATHSCI, PASCAL and AI TURING, jointly.

ica, the country where the genetic algorithm concept originated and still appears to enjoy considerably more adherence in distinct areas.

Notwithstanding, it turns out that developments in the science of genetic algorithms have recently exceeded some critical point, as they have quite suddenly been gaining grounds dramatically, worldwide (Fig. 1). Other strong indicators of the recognition of genetic algorithms science exist, related to these developments. For instance, the progenitor of genetic algorithms, J.H. Holland, has received one of the 1992 MacArthur Genius Awards. (This prestigious type of fellowship was also awarded to D.E. Rumelhart for his work on artificial neural networks a few years ago.) Furthermore, the first issue of a new journal — *Evolutionary Computation* — is scheduled for early 1993, according to a preliminary announcement of its editor-in-chief, K.A. De Jong. The aim of this journal is to provide an international forum for facilitating and enhancing the exchange of information among researchers involved in both the theoretical and practical aspects of computational systems of an evolutionary nature. As the worldwide upsurge in distinct fields of application — recently including applications in computational chemistry — seems to continue, genetic algorithms are increasingly likely to realize, in the longer term, the principle that they themselves so strongly advocate: survival.

In writing this tutorial we aimed to 'separate the chaff from the wheat' for the sake of perspective and clarity. In particular, we focus primarily on fairly well-established mainstream issues that are of relevance for comprehension of principle and for application. We avoid issues that seem to enjoy little adherence or which are still the subject of substantial debate and controversy between leading genetic algorithm scientists. Furthermore, we dissociate ourselves from the kind of exaggerated optimism and euphoria that often seems to mark the beginning of a promising new science, as has been the case with, e.g., expert systems in the early eighties and with artificial neural networks in the mid-eighties.

Finally, we emphasize that our paper truly is a tutorial in the first place. That is to say, its emphasis lies predominantly on explaining and making explicit the general principles and methodology, not on a detailed overview and discussion of specific applications (although pointers towards the latter are provided). Altogether, it is hoped that this tutorial will bring genetic algorithms science closer within the reach of a broader public of computational chemists.

1. INTRODUCTION

Genetic algorithms are among so-called simulated evolution methods — problem solving methods that simulate a natural evolution process. As an apposite way to directly hit upon the central idea behind genetic algorithms, we cite the following persuasive philosophical observations made by the English scientist Richard Dawkins on natural evolution [3]:

> Living things are too improbable and to beautifully 'designed' to have come into existence by chance. How, then, did they come into existence? The answer, Darwin's answer, is by gradual, step-by-step transformation from simple beginnings, from primordial entities sufficiently simple to have come into existence by chance.

> Each successive change in the gradual evolutionary process was simple enough, relative to its predecessor, to have arisen by chance. But the whole sequence of cumulative steps constitutes anything but a chance process, when you consider the complexity of the final end-product relative to the original starting point. The cumulative process is directed by non-random survival.

In the course of this paper we will elaborate extensively on this principle of selective incremental build-up — a process of cooperative random events with a fundamentally non-random impact, perhaps paradoxically. It cannot be denied, with so many lively precedents around us, that this simple principle — based on Darwin's classical rules about natural evolution: 'struggle for life' (competition rule) and 'survival of the fittest' (selection rule) [4] — has managed to accomplish amazingly complex natural design tasks. Certainly, it has inspired and motivated several scientists to simulate the process on the computer in order to accomplish complex computational tasks. One of these scientists was John H. Holland at the University of Michigan — the pioneer of genetic algorithm science in the early sixties. Others have pioneered simulated evolution methods in different ways, which are briefly passed in review in Section 8.1.

The mechanics of evolutionary problem solving can be summarized as follows. Any mimicked evolution process used for problem solving constitutes generalized evolutionary components such as: a population comprising artificial creatures, competition between these creatures, reproduction to create new creatures who replace the creatures that have reached the end of their life-time, semi-random changes that might lead to fitter creatures, etc. The artificial creatures in such a simulated evolution process are computer-coded candidate solutions to a complex problem. Without prior knowledge about the problem, these candidate solutions are initially arbitrary. They are, however, updated step-by-step by the separate procedures in the simulated evolutionary process. These procedures are guided by a master procedure, which acts as an arbiter who judges the relative quality of the candidate solutions, and as such plays the role of an artificial environment wherein some may survive while others will not. Altogether, the interplay between all these components expectedly leads to gradual improvement, which may ultimately culminate in a solution to the problem.

It is important to realize that in simulating an evolution process, one is not confined by the so-called Panglossian assumption, which states that what nature does must be the best thing to do. Indeed, the primary purpose of simulating an evolution process, and of genetic algorithms in particular, is to tackle complex problems and not to model evolution itself. Therefore, one merely borrows those principles from natural evolution that are considered useful or convenient for some reason, e.g. for search efficiency or for ease of implementation. Moreover, genetic algorithms conform only loosely with the borrowed principles of population genetics. That is, they are not fixed tools within whose limits an application must be written; they should in fact be regarded as environments that can be adjusted to match the task at hand. While the expert genetic algorithm scientist usually considers all this configurational freedom as an advantage and often even a necessity, the novice in the field is easily confused and overwhelmed for the same reason. For this reason, the novice practitioner would probably most appreciate an outline which emphasizes 'unifying', fundamental concepts explained in a simple, direct, and intuitively appealing way. This is precisely what we pursue throughout this tutorial. Additionally, we also highly recommend the textbook of Goldberg [2], the tutorial of Davis [5], the popular review of Holland [6] and the textbook of Michalewicz [7].

2. A BRIEF ENCOUNTER WITH APPLICATION

An appreciation of the versatility and potential of genetic algorithms is gained with a brief review of some applications.

2.1. Application in general

Although applications of genetic algorithms are spread over numerous sciences, it turns out

that many applications of practical importance can be categorized into one of the following three problem classes:

NUM: Numerical parameter estimation
SEQ: Sequencing
SUB: Subset selection

Problems in problem class NUM often involve mechanistic models, or calculation models. Such models describe a relationship between input data — collected from a system — to values for the system parameters. Calculation models are frequently encountered in engineering sciences. Examples are: economic models, environmental models, ecological models, etc.

Preferably, one would like to use a calculation model in a deductive mode: starting from the input data, deduce values for the system parameters in a mathematical–analytical way. However, since non-linearly interacting components are often present, a deductive approach is often not feasible in practice. An alternative, then, would be to use the calculation model the other way around: propose a candidate solution and see how well the input data are approached according to some optimality criterion. This demands an iterative approach, called optimization. The principles of optimization will be outlined in some detail in Section 3.

Optimization methods can be categorized according to the degree to which they make assumptions about the problem. Optimization methods that make few assumptions about the problem are called weak. They can be widely applied, but also take a relatively long time to find an acceptable solution. Strong optimization methods, on the other hand, make many assumptions about the problem. They converge quickly and are limited to a small range of relatively simple problems. When applied to a problem outside this range, they tend to find false solutions or even break down completely.

Genetic algorithms take a standpoint between weak and strong optimization methods — they are moderate. Moderate methods may be competitive under circumstances wherein both weak and strong methods are likely to fail. For instance, when a calculation model for a NUM problem is increasingly complex, strong methods will increasingly fail as the assumptions they are based on collapse. When, in addition, the problem dimensionality is high, weak optimization methods may require too much time, given practical time constraints. Under such real-world circumstances, moderate methods — genetic algorithms in particular — typically become attractive choices. This emphasizes an important general point about genetic algorithms: they are not a panacea, but make applicable sense only for complex, large-scale problems.

Problems in problem classes SEQ and SUB belong to the more general class of combinatorial problems. Examples of problems in SEQ are scheduling, capacity planning, inventory problems, routing problems, etc. Examples of problems in SUB are feature selection problems, partitional clustering, etc. Like real-world problems in NUM, real-world problems in SEQ and SUB too are often not mathematical–analytically solvable, they are very complex, and high-dimensional. In this case, again, genetic algorithms are likely to comprise better choices than either very weak or very strong optimization methods.

To give an impression of the versatility of genetic algorithms, we mention just a few areas where applications are reported:

- image processing, e.g. of medical pictures;
- pattern recognition, e.g. for feature selection;
- forecasting, e.g. of weather;
- modeling and system identification, e.g. with splines;
- adaptive filtering, e.g. for noise canceling;
- robotics, e.g. for trajectory generation;
- network design, e.g. for computers and integrated circuits;
- design of bearing constructions, e.g. of bridges;
- operations research, e.g. for job scheduling and capacity planning;
- adaptive document retrieval, e.g. for libraries.

For a more elaborate survey of applications we refer the reader to ref. 2.

2.2. Application in chemistry: an overview

Genetic algorithms were recently introduced to chemists [8]. Various computational-chemical

applications were reported later. As it is beyond the purposes of this tutorial to treat applications, we merely provide an overview with some pointers to the literature (unreferenced work is in preparation for publication):

- Conformational analysis of biopolymers (NUM problem, by Blommers et al. [9], Levy et al. [10] and Lucasius et al. [11–13]
- Training of neural networks for interpretation of X-ray fluorescence spectra (NUM problem, by Bos et al. [14])
- Wavelength selection in multi-component analysis (SUB problem, by Lucasius et al. [15])
- Flowshop scheduling in chemical engineering (SEQ problem, by Cartwright et al. [16])
- Design of synthetic reaction networks (SEQ problem, by Fontain et al. [17])
- Inversion of neural networks for the analysis of industrial process conditions (NUM problem, by De Weijer et al. [18])
- Multi-criteria optimization in atomic emission spectroscopy (NUM problem, by Wienke et al. [19])
- Prediction of protein tertiary structure (NUM problem, by Schulze-Kremer [20])
- Sequential assignment of NMR-spectra of proteins (SUB/SEQ problem, by Wehrens et al. [21–23])
- Multi-criteria optimization of photometric calibration curves (NUM problem, by Wienke et al. [24])
- Optimization of calibration data by wavelength selection (SUB problem, by Li et al. [25])
- Prediction of fragmental retention indices in HPLC analysis (NUM problem, by Wehrens et al.)
- Curve fitting of spectra of PET and PEN yarns (NUM problem, by De Weijer and co-workers [26,27])
- Partitional clustering (SUB problem, by Lucasius et al.)
- Non-linear variate compaction (NUM problem, by Lucasius et al.)
- Multivariate regression based on fuzzy theory and with least median of squares (NUM problem, by Rajkó et al.)
- Non-linear modeling of chromatographic behavior (NUM problem, by Lopes Marques et al. [28])

There also exists work of computational-chemical interest that is related to a simulated evolution methodology which is different from genetic algorithms. We mention the work of Eigen [29–31], Schuster [32] and Wang [33] on molecular evolution (SEQ problem). Further, Wu et al. [34,35] have determined NMR pulse shapes (NUM problem). Finally, Güntert et al. [36] add an evolutionary dimension to their program DIANA for the conformational analysis of biopolymers (NUM problem).

From the first results obtained in applying simulated evolution methods, and genetic algorithms in particular, to computational-chemical problems, it is concluded that the methodology appears to have considerable potential in complex large-scale problem solving. In order to understand these and other successes achieved, the remainder of Part 1 of this tutorial is devoted to a general discussion of the principles and properties of genetic algorithms and their wider context. Starting from that base, Part 2 [37] is devoted to the practicalities of the methodology.

3. EMBEDDING GENETIC ALGORITHMS IN OPTIMIZATION SCIENCE

In this section we will discuss some basics of optimization [38–41] inasmuch as they are relevant to the understanding and application of genetic algorithms. Three purposes are thereby served. First, we treat material which is often considered to be presupposed knowledge in the literature on genetic algorithms. Thereby we aim at facilitating access to the literature. Related mathematical rigor is circumvented. Second, by placing genetic algorithms within the more general embedding of optimization science, their specific properties can be better validated and their mechanics better appreciated. Third, it turns out that there are theoretical limits to problem tractability, which any practitioner involved in optimization should be aware of. That is, the efficiency of any method, no matter how powerful

it is claimed to be, is ultimately limited by the complexity of the problem itself.

Occasionally we will, for illustration purposes, refer to three simplified versions of a problem that concerns optimizing retention (selectivity) in HPLC analyses. Important factors in this problem are assumed to be the composition of the mobile phase and the column type. Two mobile phase components, A and B, are considered and their concentrations are indicated as c_A and c_B. The column type is indicated as γ. We instantiate the following three versions of the problem:

- Problem HPLC1:
  For fixed γ, what are the optimal c_A and c_B?
- Problem HPLC2:
  For fixed c_A and c_B, what is the optimal γ?
- Problem HPLC3:
  What are the optimal c_A, c_B and γ?

3.1. Problem parameters and the problem space

Basically, any problem can be defined as a general question to be answered, usually involving several variables, or problem parameters (e.g. c_A, c_B and γ in Problem HPLC3), which are identified by the creator of the problem solver and whose values are initially unspecified. For D problem parameters, the problem is said to have dimensionality D.

The nature of the problem parameters follows from the problem description. For some problem parameters the values may be on a qualitative scale (e.g. γ); for others they may be on a quantitative scale (e.g. c_A and c_B). Each problem parameter has a set of legal values associated with it — values that cannot a priori be ruled out according to the creator of the problem solver. Any combination of legal values for the problem parameters represents a candidate solution. It is often convenient to represent a candidate solution as a vector, or string, of which each element is a legal value of the corresponding problem parameter (Fig. 2). In genetic algorithms parlance, the string representation of a candidate solution is often called a 'chromosome' or 'genotype' (genetic blueprint), by biological analogy. We will, however, continue to use the more neutral term 'string'. We also avoid some other tendentious terms, such as 'gene' (problem parameter) and 'allele' (value for a problem parameter).

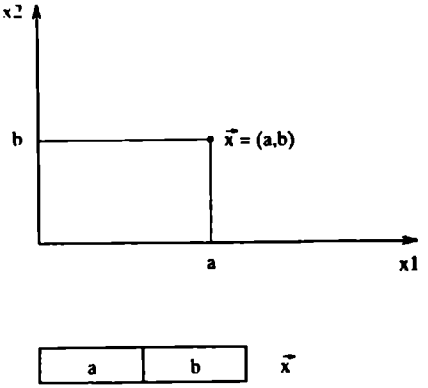

Fig. 2. A candidate solution for a problem that involves two problem parameters may be represented as a two-dimensional vector or string (x). Here the collection of candidate solutions comprises a two-dimensional Euclidean problem space.

The collection of all candidate solutions comprises the problem space. For instance, in the special case of continuous numerical problem parameters, such as in Problem HPLC1, we have a Euclidean problem space in which each candidate solution can be represented as a point (Fig. 2). In general, only one or at most a subset among the entire set of candidate solutions represents the true solution, and finding it is, of course, the purpose of problem solving. For our convencience, we shall from here on understand by 'the' true solution all true solutions that the problem can have.

3.2. Problem representation

The creator or the problem solver has the freedom, in principle, to choose any representation considered useful or convenient for the problem parameter values. For instance, he/she may want to use auto-scaled values for quantitative problem parameters, say for c_A and c_B in Problem HPLC1. In other situations it may be a better choice to use an integer representation for c_A and c_B instead of real values, e.g. when there is only a limited number of 'standard' values from which one can choose in practice. In that case, it might be a wise choice to attach a quantitative meaning to the integer values as well. In particu-

lar, the mapping from integer ('encoded') values to real ('decoded') values could be defined in such a way that increasing integer values map to increasing real values. (We will come back to such integer–real mappings more elaborately in Part 2 [37].) For the column type γ, too, an integer representation could be used as an encoding, but then the integer values would have no quantitative meaning.

A basic expressional device, or alphabet, is required for any representation. The number of characters in an alphabet is constant and referred to as the alphabet's cardinality. Familiar examples are the roman alphabet, which is 26-cardinal, and the alphabet used by nature to encode life forms, which is 4-cardinal. Problem parameters are usually coded in terms of alphabets with relatively small cardinality. For instance, the value of an integer-coded problem parameter, say 413, may be regarded as a word composed by characters copied from the 10-cardinal alphabet $\mathscr{A}_{10} = \{0,1,2,3,4,5,6,7,8,9\}$. Likewise, the value of a real-coded problem parameter, say 58.39, could be regarded as a word composed by characters copied from the 11-cardinal alphabet $\mathscr{A}_{11} = \{0,1,2,3,4,5,6,7,8,9,\cdot\}$. In general, the encoding or decoding of problem parameter values basically involves a translation from one alphabet to the other.

The role of alphabets is strongly emphasized in the literature on genetic algorithms. Since any alphabet with a cardinality larger than or equal to 2 can be used to encode any kind of information, it is perhaps not surprising that there is currently still much debate about which alphabet is best to use under which circumstances [42]. In this tutorial, too, the question of when to use which encoding is sometimes addressed.

3.3. Objective functions

In general, a problem solver is a step-by-step strategy, or procedure, or method, intended for problem solving. Direct and indirect methods are commonly distinguished. A direct method is deductive, i.e. it starts directly from the input data and works its way towards the true solution through mathematical-analytical steps (e.g. logical or calculus-based steps). However, when a problem is 'ill-behaved' (which might be the case when it is nonlinear or discontinuous) such that direct approaches are not feasible, one will need to resort to indirect methods. These comprise the class of problem solvers to which genetic algorithms belong.

Indirect methods are iterative procedures that make a sequence of supposedly improving guesses at the true solution in the problem space by guided trial and error. This can only be accomplished by way of an optimality criterion or evaluation criterion — an arbiter that judges each guessed solution (candidate solution) for its performance — which must thus be predefined. Perhaps counterintuitively, very complex problems can often be approached with a strikingly simple evaluation criterion. For example, the archetypical traveling salesperson problem — 'what is the shortest n-city circular path?' (with n an integer constant) — has been proved to be extremely complex on theoretical grounds, yet the evaluation criterion merely needs to sum city-to-city distances in a candidate path in order to pass its judgment on performance. (The complexity of this problem is due to *NP*-completeness; see Section 3.8.2 for a discussion of this property.)

An evaluation criterion is implemented as an objective function, ϕ, which is generally defined by the relationship:

$$y = \phi(x)$$

where $x = (x_1, x_2, \ldots, x_D)$ is the string representation of a candidate solution for a problem with D problem parameters, and $y = (y_1, y_2, \ldots, y_N)$ is a string of N scalar performance parameters. Vectorized performance is important in multi-criteria problem solving, but we confine this tutorial to mono-criterion problem solving, with no loss of purpose. For instance, in Problem HPLC3 the objective function can be defined as:

$$y = \phi(c_A, c_B, \gamma)$$

where y is a scalar. Hence, in Problem HPLC3 the search for the optimal HPLC configuration basically comes down to finding an optimal value for y: the maximum value of y if it is defined as utility, merit, quality, score, etc. — the minimum

value of y if it is defined as cost, error, loss, etc. Since any minimization problem can always be transformed into an equivalent maximization problem, we will treat indirect problem solving — or henceforth simply optimization — from now on as maximization without loss of generality.

This is convenient because genetic algorithms always maximize the performance parameter, or the 'fitness', or 'phenotype', by biological analogy. Later, we will explain in more detail how, in genetic algorithms, the objective function plays the role of a simulated environment in which strings with a higher assigned fitness will be more likely to pass their code (information content) to future strings.

3.4. Hill-climbing on landscapes

In optimization, the problem space is called the search space, S. Each evaluation made during the optimization procedure may be construed as a 'measurement', or observation, of the performance parameter y conducted in the search space. Accordingly, the combination of the search space and the objective function results in an observation landscape, or just o-landscape for brevity (Fig. 3). Optimization thus comes down to 'hill-climbing' on the o-landscape, with the intent of finding the highest peak. The latter is called the global optimum, and the candidate solution it refers to is considered the true solution — also called the global solution in this context. All optima that are not a global optimum are called sub-optima or local optima, and the candidate solution that corresponds with a local optimum is called a local solution.

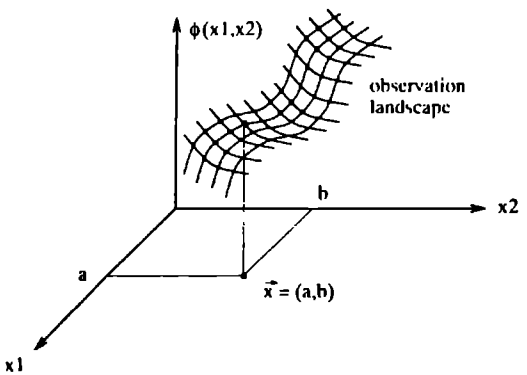

Fig. 3. The combination of the objective function and the search space results in an observation landscape.

The chain of guesses made at the true solution during optimization may be construed as a search trajectory — the path that is traversed through the search space. If there is no way of verifying whether the global optimum has been reached upon terminating the search (which is usually the case in real-world problem solving), then the 'best-so-far' solution obtained in the search trajectory is called the estimated solution, which may not be the true solution but still 'good enough'. In that case, too, the problem is often said to be solved, although, strictly speaking, of course it is not.

Genetic algorithms distinguish themselves from many other optimization methods in that they work with a population of string-represented candidate solutions. Hence, since population-to-population steps are made instead of point-to-point steps, optimization with genetic algorithms may be pictured as a swarm of points that roam along separate search trajectories over the o-landscape in search of the global optimum. Intuitively, conducting several searches effectively in parallel constitutes a more robust strategy. In addition, it can be shown on theoretical grounds (Section 5), that genetic algorithms are highly efficient, as the information processed along one search trajectory is used in a meaningful way along other search trajectories for guidance.

3.5. On-line and off-line optimization

Time is required, of course, to evaluate a candidate solution, i.e. to conduct a 'measurement' of performance in the search space. If an actual (real-time) measurement, e.g. an instrumental measurement, is part of the evaluation, then the evaluation criterion and the optimization procedure that it is part of are called on-line. Otherwise, when only computation is required, these are called off-line. Computation is usually much faster than instrumental measurement.

An example of an optimization method which is often applied on-line is the simplex method. On-line simplex optimization may, for instance, be used to tackle Problem HPLC1, whereby the evaluation of each candidate solution would involve instrumental measurement of retentions. On the other hand, however, if an appropriate calculation model for the retention behavior were available, then there would be no need to conduct laboratory measurements, and optimization could be carried out off-line by any suitable method. In passing, it should be realized that if the calculation model could be solved directly, there would probably be no point in using optimization at all.

The distinction between on-line and off-line evaluation or optimization, as the case may be, is drawn because many problems are so large and complex that even very efficient on-line optimization cannot be expected to converge within practical time limits. For instance, if finding the global solution in a search space of, say, 10^{100} candidate solutions requires as 'little' as, say, 10^{10} evaluations (on the average), then on-line optimization will nearly always be prohibitive, despite the appreciable efficiency, 10^{90}, of the method applied. Such situations are typical of search by genetic algorithms, which is why they usually demand off-line evaluation criteria.

3.6. Search heuristics

In order to have a reasonable chance of finding the global optimum, the creator or the problem solver must implement some mechanism or another that makes 'intelligent' steps on the o-landscape, such that it is likely that jumps are made towards better performing regions — be it either in the short or in the long term. Such provisions for guidance are called transition rules or search heuristics.

Any search heuristic requires some minimal step size, or progression unit, to be observed in one way or another such that the search space becomes finite. In general, a larger minimal step size ensures quicker convergence, but possibly entails a higher risk of ending up in a local optimum. Further, one may distinguish deterministic and stochastic search heuristics. The former are reproducible, while the latter are based on random events. As will become evident in due course, genetic algorithms employ stochastic search heuristics, comprising an interplay between several elementary stochastic operators.

In effect, any search heuristic by definition makes implicit assumptions about the structure or shape of the o-landscape, i.e. about the 'regularities' inherent to the problem. The most generally applicable optimization methods make only one, the weakest possible, assumption about the shape of the o-landscape, namely: that the objective function is defined for each candidate solution. Examples of such methods are:

- *Enumerative search*: systematically scans the search space and in each time step remembers the 'best-so-far' candidate solution on the search trajectory, e.g. grid search in an Euclidean search space. These search heuristics are called exhaustive.
- *Monte Carlo search*: conducts an unbiased random walk through the search space and in each time step remembers the 'best-so-far' candidate solution on the search trajectory. These search heuristics are called blind.

Exhaustive and blind search heuristics are examples of so-called naive search heuristics. A naive search method is interesting in that its inefficiency, i.e. the expected number of time steps that are required to reach the true solution, can easily be derived; namely, the size of the search

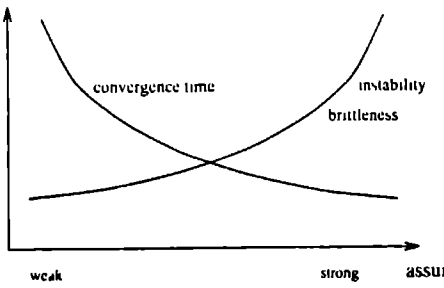

Fig. 4. Commonly observed trade-off in applied optimization: with increasing assumptions in the method, convergence time decreases, but instability in results increases and brittleness of the method increases.

space. This number can be used as a 'worst-case' or 'most inefficient' reference relative to which an experimentally estimated (in)efficiency of another search method may be judged. Comparisons of this kind are often made in genetic algorithm research.

Methods which make weak implicit assumptions about the o-landscape are called weak methods, or general methods. On the other hand, methods which make strong implicit assumptions about the o-landscape are called strong methods, or dedicated, or tailored methods. In order to understand the importance of the place that genetic algorithms occupy in the 'general-tailored methods spectrum', it makes sense first to consider the properties of methods near the boundaries in this spectrum (Fig. 4).

As a rule, the wide applicability of general optimization methods is overshadowed by a high inefficiency. For instance, in the extreme case of Euclidean grid search the whole search space is scanned. In this example, when we take, say, L levels for each of the, say, D problem parameters, then the size of the search space is L^D — it is exponentially dependent on D. (For $L = 1000$ and $D = 25$, which are not unrealistic values, this comes down to 10^{75} — about the estimated number of atoms in the universe!) Hence, with increasing D, any method which employs naive search heuristics is evidently bound very soon to lose the battle against the so-called 'curse of dimensionality'.

Tailored optimization methods, on the other hand, tend to find an optimum very quickly as their search heuristic only considers a local area on the o-landscape. Such search heuristics are called local, or greedy, search heuristics. Among widely used greedy methods are gradient search and simplex optimization. The former assumes that the derivative of the o-landscape exists and is continuous, whereas the latter assumes certain geometrical properties of the o-landscape, such as the enablement of inflections. These assumptions limit their applicability. For these reasons, tailored methods are called fragile, or brittle [43]. For instance, although gradient search and simplex optimization might work well for Problem HPLC1, they break down immediately on the 'rugged' o-landscapes of Problems HPLC2 and HPLC3 since the latter two problems both involve a qualitative problem parameter (γ). Even when tailored optimization methods do not break down on a particular problem, another disadvantage is still that they find the closest optimum relative to the starting point in the search space: the apparent gain in efficiency appears to be overshadowed by a loss in reproducibility, i.e. tailored methods are not stable, not robust, and not efficacious.

Somewhere about midway between the very general methods and the very tailored methods in the general–tailored methods spectrum, genetic algorithms fill an important niche by compromising between the advantages of both extremes with remarkably little sacrifice. In a methodological sense, genetic algorithms are neither weak nor strong, but moderate. Among other moderate optimization methods are other simulated evolution methods and simulated annealing (see Section 8.1). It is not surprising that many moderate methods employ stochastic search heuristics. This is because the use of random events provides a graceful means of tuning the proportion of exploration (due to uniform random events) to exploitation (due to deterministic and/or distributed random events). Put differently, the strength of assumptions about the problem — these reside in the exploitational part of the search heuristics — can be tuned to the desired level.

3.7. Apparent and inherent problem dimensionality

Saying that a method is more efficient for a particular problem than naive search is, is essentially the same as saying that the effective, or apparent, size (or dimensionality) of the problem's search space is smaller than the formal size (or dimensionality). The apparent problem dimensionality can only be estimated experimentally for a problem of which the global optimum is known in advance; namely, by replicating a number of runs, then recording in each of them the number of evaluations required to hit the global optimum, and finally to take the average of these numbers as the estimate. Nevertheless, the concept of 'ap-

parent problem dimensionality' is used for all problems, including those of which the global optimum is not known in advance

When the apparent problem dimensionality, d, is smaller than the formal problem dimensionality, D, then this is an indication that regularities in the o-landscape have been exploited by the search heuristics such that supposedly unpromising parts of the formal search space could be skipped, i.e. that the search method has been 'learning' In this respect, the reader may draw a loose comparison with pattern recognition techniques (dimension reduction techniques), such as variable selection or latent variable creation Note that, since different search methods perform differently, in general, on the same problem, d must depend on the search method used. Quite naturally this leads us to state the following trivial principle, which we will, for the sake of brevity, refer to as the

Principle of optimal method. Given a problem, then some problem representation and some search heuristics must exist which, when combined, lead to the most efficient feasible search method, i.e to the lowest feasible apparent problem dimensionality.

This absolutely minimal apparent problem dimensionality is called the inherent problem dimensionality — 'inherent' because it may be considered a pure property of the problem However, this property is hypothetical — i e it cannot be determined in practice — because the number of possible problem representations and the number of possible search heuristics are infinite, and there is no general methodology available to find the optimal combination.

In Section 4 2 we will rephrase the principle of optimal method in the context of genetic algorithms.

3 8 Inherent problem complexity and tractability

The importance of the inherent problem dimensionality is merely that it serves as a theoretical bound to the efficiency that can be attained by any search method, i.e. as a measure of inherent problem complexity The more complex a problem is, i.e the higher its inherent dimensionality, the less tractable it is said to be.

Upon approaching complex, large-scale optimization problems, it is important to be aware of theoretical limits to problem tractability as dictated by inherent problem complexity We briefly review three established lines of approach for the qualification or quantification of problem complexity

3 8 1 Empirical approach

The first approach to problem complexity is empirical [44–48] It aims at experimentally quantifying the structure of the o-landscape. This can be done by way of a random walk through the search space with some finite step size From the time-series thereby obtained for the performance parameter — assuming constant statistical properties over the whole o-landscape — one can derive the correlation distance A large value for the correlation distance, then, indicates a 'smooth' o-landscape (i e an apparently simple problem), whereas a small value indicates a 'rugged' o-landscape (i e an apparently complex problem) In passing, note that, if there is no 'smooth' path at all leading to the global optimum, no search heuristics can be expected to perform better than naive search heuristics

The advantage of empirical complexity problem analysis is that the approach is universal and therefore provides a useful aid in the design of an optimization method That is, the designer will measure the apparent problem complexity for several combinations of problem representation and search heuristics, and the one which yields the apparently smoothest o-landscape is eligible

3 8 2 Theoretical approach

The second approach to problem complexity is primarily useful as an attitude for the practitioner, since otherwise it makes only theoretical sense [39]. It addresses the theoretical question of how the inherent size of the search space — which, as argued before, cannot be determined in practice — would increase with increasing problem dimensionality D, i e upon problem scale-up It follows that it makes sense to distinguish two

problem classes, P and NP. P is the class of problems of which the inherent size of the search space increases polynomially with D (as a polynomial of finite order). NP is the class of problems of which the inherent size of the search space — when the search is carried out by a method that uses a non-bounded number of processors in parallel — increases polynomially with D. For problems in class P it is believed that it is relatively easy to find a method such that the apparent size of the search space grows polynomially (slowly) upon scale-up. These may be regarded as simple problems. On the other hand, for problems in class NP, it is believed that it is relatively difficult to find a method such that the apparent size of the search space grows polynomially upon scale-up, i.e. it is believed that this growth will often be non-polynomial (exponential, explosive). The provably hardest problem in class NP is the archetypical satisfiability problem, and every problem that can be formally proved equivalent to it is called NP-complete. Even though some problems can be proved to be NP-complete, when a particular method is applied to it there is, in general, no way of knowing in advance whether the apparent size of the search will grow polynomially or non-polynomially upon scale-up. Assuming that such growth could be estimated or measured (which is questionable because scaling up is not a uniquely defined procedure for all problems), then in the former case the method is said to show polynomial time behavior and in the latter case it is said to show exponential time behavior, for the particular problem (Fig. 5). In practice, polynomial time behavior often leads to finding the true (or a good enough) solution within 'reasonable' time — that is, within practical time constraints of, say, an hour, a day, a week, a month, or a year, but typically not a million human lifetimes.

Since, on many occasions, genetic algorithms turn out to compare favorably with respect to other methods in solving provably NP-complete large-scale problems within reasonable time, many experts believe that genetic algorithms are among the methods that most frequently solve problems in polynomial time. A frequently used argument for this belief is theoretical evidence

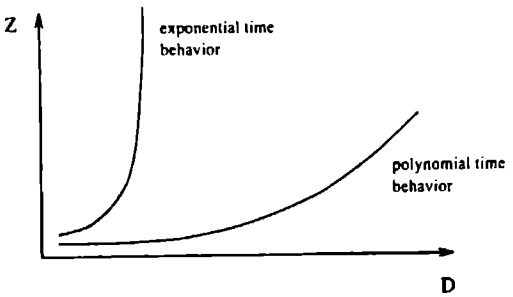

Fig. 5. The apparent size, Z, of the search space may increase in polynomial or in exponential dependency on the problem dimensionality, D.

that the search behavior of genetic algorithms is exponential (Section 5).

3.8.3. Causal approach

The third approach to problem complexity is causal [2,49,50]. It explains problem complexity in terms of epistasis (interaction, correlation, nonlinearity) between the problem parameters, as can be defined by the following inequality about the objective function:

$$\begin{aligned}&\phi(\ldots,x_i,\ldots,x_j,\ldots)\\&\quad-\phi(\ldots,x_i',\ldots,x_j,\ldots)\\&\neq\phi(\ldots,x_i,\ldots,x_j',\ldots)\\&\quad-\phi(\ldots,x_i',\ldots,x_j',\ldots)\end{aligned}\qquad(1)$$

Stated in words, the change in the value returned by the objective function upon a change in the value of one problem parameter (x_i) depends on the current value of another problem parameter (x_j). A higher level of epistasis is seen as a cause of increased problem complexity.

The notion of epistasis is important in the discussion of Holland's mathematical framework for genetic algorithms: the schema theorem (Section 5).

4. OPERATION

The central data structure in genetic algorithms is a (usually fixed-size) population of

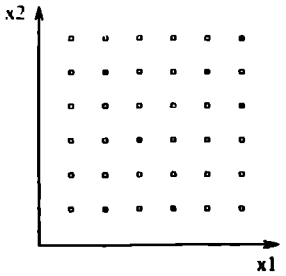

Fig. 6. Illustration of a two-dimensional discrete Euclidean search space: solid and open squares. Example of a population as a sample of the search space: solid squares.

string-represented candidate solutions. In a statistical sense, the population is a sample of the search space (Fig. 6). Accordingly, the search space is sometimes also called the grand population.

Numerous types of representation, as well as search heuristics by which the population can be updated in each time step, or generation, are currently in use. For the present purposes, we will confine ourselves to simple representations and simple search heuristics.

4.1. A simple representation

In many applications, the candidate solutions that comprise the population are encoded as bitstrings, composed by characters copied from the binary (2-cardinal) alphabet {0,1}. This choice is due in part to the fact that, earlier, Holland [51] used this alphabet in his theoretical work on genetic algorithms. However, a binary representation, although by its nature convenient for implementation on computers, is not necessarily the most efficient one for all problems; this follows immediately from the abovementioned principle of optimal method (Section 3.7). We, too, will focus regularly on binary representations without loss of generality wherever it is convenient for the discussion.

In general, all bitstrings in the population have the same format. A bitstring format prescribes how a bitstring is sub-divided into contiguously placed binary words, or bitfields (Fig. 7). This implies that all bitstrings in the population have the same length. Each bitfield corresponds with a problem parameter and has a minimum length of one bit as it must carry information.

Albeit practicalities concerning representation are outlined in Part 2 [37], we believe the further discussion would profit by the following brief illustration of bitfield decoding.

Example. Suppose a bitfield consists of 3 bits and has the base-2 (binary) value 101. Now if the purpose of decoding is to obtain a base-10 (decimal) value for the problem parameter that the bitfield applies to, then one way to do that might be to simply take the integer interpretation of the bitfield. This, then, will yield the desired decimal value: $D = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$. Note that, in general, a B-bits bitfield can take 2^B discrete values, which for $B = 3$ is 8.

In another vein, suppose the purpose of decoding is to obtain a real value for the problem parameter that the bitfield applies to. Then one can divide some confined range $[v, w]$ that any possible value is assumed to lie in, into 2^B equidistant levels (quanta). After the aforementioned integer interpretation of the bitfield, yielding decimal value D, the Dth level specifies the decoded real value R:

$$R = v + (w - v) \cdot \frac{D}{2^B}$$

which for bitfield 101 is $v + (w - v) \cdot 5/8$.

4.2. Discussion of a flowchart using simple genetic operators

When the values of the problem parameters obtained after decoding a bitstring are passed to the objective function, this effectuates the calculation of a fitness value that will be associated

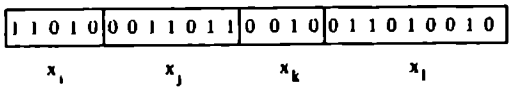

Fig. 7. Example of the binary representation of a candidate solution that involves problem parameters x_i, x_j, x_k and x_l for which the bitfield sizes are specified by bitstring format '5 7 4 9'.

with the bitstring (by means of simple bookkeeping). As pointed out earlier, the search heuristics of any genetic algorithm use the fitness values in some way or another to encourage competition between the bitstrings for reproduction (replication), followed by semi-random changes. Such an update is done periodically, whereby the population evolves dynamically in a step-by-step fashion, in much the same way as natural life-forms in an ecological population do. The ultimate aim of this is the dissemination of improved genetic code, or information, either in the natural or artificial environment.

We feel that, despite the lack of a standard flowchart in applications of genetic algorithms, the following simple and traditionally widely adopted scheme (Fig. 8) conveys the general concepts reasonably well.

1. Initiate

Usually, the initial population in a genetic algorithm is selected in a random way, i.e. the bits of the bitstrings are assigned random values. This may be done either to minimize bias or to maximize the number of possible recombinations for the recombination operator (discussed in Module 4), or both. Maximizing the number of possible recombinations is another way of saying that the search space is optimally 'spanned'.

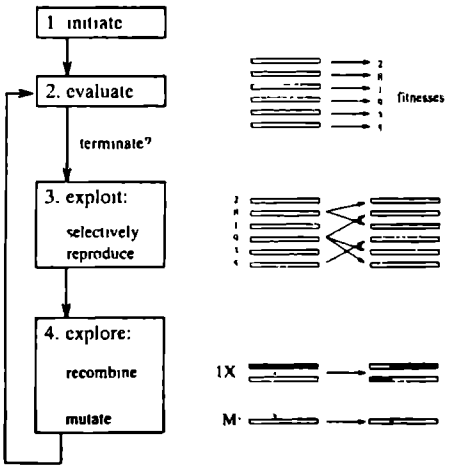

Fig. 8. Flow chart of a simple genetic algorithm.

Sometimes, however, it may be convenient to choose the initial population heuristically rather than randomly, so that the genetic algorithm does not search a space that is unnecessarily large [52]. The size, N_p, of the population is commonly chosen somewhere in the range of [50, 500] members for most practical applications.

2. Evaluate and Terminate?

For each bitstring, the fitness is evaluated by calling the objective function. Fitness evaluation is usually the step with the highest computational load, i.e. it determines the overall speed of execution. The objective function is often said to comprise the domain-dependent module of a genetic algorithm, i.e. all other modules are not 'aware', in principle, of the kind of problem that they are dealing with. Accordingly, the latter modules are commonly referred to as domain-independent, although strictly speaking, they are domain-dependent in the sense that the genetic algorithm designer is usually inclined to customize any module to the nature of the problem if that is somehow beneficial, as pointed out above.

Next, a termination criterion is applied, which is usually based on some convergence measure. If it fails, the population must be adjusted by using search heuristics such that it is likely to improve — otherwise execution is aborted. Alternatively, the termination criterion may be simply based on some imposed maximum generation.

3. Exploit: compete and selectively reproduce

The first step towards populational reorganization is selective reproduction. A new (equally sized) population is created from the current population, and upon completion the latter is replaced. For a population size N_p, this procedure starts with N_p times performance of the task: 'indicate a bitstring in the population according to a predefined selection criterion, then set aside a reproduction (a copy) of it for the new population to be formed'. Replacement of the population by the new population after completion of this iterative procedure implies that individual bitstrings have a limited lifespan.

When the selection criterion shows bias for the higher ranking bitstrings in some way or another,

the updated population may be expected to perform better than its predecessor, on the average. A commonly used selection criterion with such filtering characteristics is one that assigns to each bitstring in the population a target sampling rate which is proportional to the bitstring's fitness. This procedure is not difficult to implement on a computer and can be compared with rolling a biased N_p-faced die, where each face on the die corresponds uniquely to a bitstring in the population, and vice versa. The die is biased in such a way that, upon rolling it, the probability that a particular face is selected is proportional to the fitness of the bitstring to which that face corresponds. Due to the probabilistic nature of 'roll a die' selection, the selected bitstrings can merely be expected to reproduce in proportion to their fitness, i.e. in practice there will be a spread in the number of reproductions for a particular string. Accordingly, the lowest performing bitstring in the population might reproduce by a small probability and/or the highest performing bitstring might not. This indicates one of the ways in which genetic algorithms can escape local optima.

Biased selection, however achieved (see Part 2 [37] for alternative ways), may be regarded as the exploitative part of evolutionary search heuristics. It is necessary but not sufficient for efficient search. It is not sufficient because the search space is not explored. Within a few generations under fitness-proportional reproduction, the population will comprise only copies of the best bitstring in the initial population. It is very unlikely that this bitstring represents the global solution, since the population is at any time only a small sample of the grand population (the search space). In general, the phenomenon whereby the fittest, yet not globally optimal, strings dominate the population is called premature convergence.

In order to counterbalance depletion of diversity in the population, genetic operators are required that introduce variation (Module 4). The imposition of variation comprises the explorative part of evolutionary search heuristics. Exploration is important, as it may lead to better performing members in the population but, on the other hand, it should not destroy too much of the useful information that has collected in the population, either. In order to deal with this trade-off, one should follow certain 'rules of the game', i.e. observe the following plausible principle, which we will, for the sake of brevity, refer to as the

Principle of maximal preservation (minimal disruption). Given an amount of variation to be imposed on the population, this should be done in such a way that there is a maximal preservation of information which has accumulated in the population during past generations and which is important for the purposes of the search task. (That is, in such a way that there is a minimal concomitant disruption of, i.e. loss in, important information.)

Since variation is by definition introduced at the level of representation, the principle of maximal preservation actually advises an optimal combination between representation and evolutionary search heuristics. Therefore, the principle of maximal preservation is essentially a rephrasing of the earlier discussed principle of optimal method (Section 3.7), in the context of genetic algorithms.

The best way to preserve information while exploring a search space depends strongly on the type of problem. Different problems often require different representations and/or evolutionary search heuristics for maximum performance. Therefore, the principle of maximal preservation is both a unifying and diversifying factor in genetic algorithm applications.

4. **Explore**

Recombine

Intuitively, it seems plausible that a constructive way to observe the principle of maximal preservation is the following. First slice a pair of bitstrings into a few fractions and then combine the latter in a different way so that two new (equally sized) bitstrings are assembled. This idea is inspired by nature's analogue: sexual reproduction. The advantage of a sexual strategy is that information in each parental fraction is preserved in the inheriting child. In this way, a lot of effort hitherto done by the evolutionary process is not lost, while major improvements may occur when the right fractions combine in making a child.

A simple recombination is Holland's classical one-point crossover, **1X**, which swaps bitsegments between two bitstrings after a common breakpoint, or crossover point, that has been selected according to a uniform random distribution (Fig. 8). In Section 5.5 we will explain, in the light of the principle of maximal preservation, under what circumstances **1X** works optimally.

Recombination introduces variation into the freshly reproduced population as follows. First the bitstrings are put randomly in pairs. Then recombination is performed iteratively on the pairs, with a probability p_r, that usually lies within the range [0.5, 1.0]. It is easily verified that, with enough diversity in the population, iterated recombination alone can reach a number of candidate solutions in the search space that is much larger than the number of candidate solutions that are actually in the population at a given point in time.

An example of lack of diversity in the population is the following. When at a given bit position, say the third, each bit value matches another on all bitstrings in the population, then this value can obviously never be changed by **1X** crossover (Fig. 9). That is, under these circumstances **1X** alone can no longer cover the entire search space. In order to ensure that recombination can cover the entire search space, it is assisted by a procedure that is called mutation, after its biological analogue:

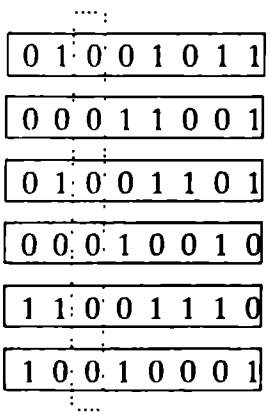

Fig. 9 Example of irrecoverable loss for 1X recombination in the population.

Mutate

In general, mutation performs local changes in the encoding throughout the population at locations selected according to a uniform random distribution. The simple mutation, M, used here for binary representation, entails inversion of the values of a small fraction of randomly selected bits in the population, i.e. a small fraction of 1-bits become 0-bits and vice versa. This mutation is applied with a relatively low probability, p_m, typically below 0.05, as its main role is to support recombination such as to protect it against irrecoverable loss. In addition, mutation maintains diversity in the population and opposes premature convergence.

Continue

This completes the 'life cycle' of the population, which leads to replacement of the current population by the new population, followed by returning to Module 2.

We conclude this section with some general comments on evolutionary search heuristics.

The variation imposed on any bitstring by exploratory operators may lead to either an increase or decrease of its fitness. Owing to the selection mechanism, any incidental increase is immediately supported and proliferated towards populations of subsequent generations, whereas any incidental decrease decays for the same reason. Thus the selection criterion provides the genetic algorithm with a self-organizing capability, as it acts as a 'marriage broker' for mutation-assisted recombination. Recombination is generally considered the most important variational operator, as can be illustrated in an intuitively appealing way by **1X** recombination (Fig. 10): by imposition of only one manipulation at the right place (breakpoint), i.e. at low cost, **1X** may combine the best performing bitstring segments into much better performing bitstrings.

Such efficient stepwise accumulation of important information reflects Dawkins's observations, which we cited in the Introduction (Section 1)

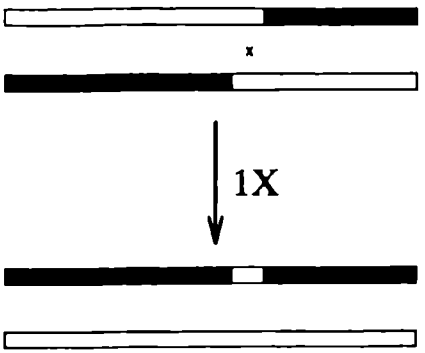

Fig 10 Example of constructive 1X recombination Black and white portions of the string indicate low and high contributions to the total string fitness, respectively The high quality child string reflects the 'discovery' of a major improvement that is likely to survive

and that we baptized the principle of selective incremental build-up At this juncture, we can now construe selective incremental build-up in genetic algorithms as a direct result of a meaningful interplay between fitness-proportional reproduction on the one hand, and recombination based on the principle of maximal preservation on the other.

Note that comparable leaps in information build-up, solely by applying mutation, are far less likely, since this would require several random bit inversions in a bitstring at the right places, in sequence Indeed, M mutation improves and destroys high-quality bitstring fractions at equal rates, i.e it lacks the favorable assembly behavior that can be achieved by recombination This is in fact why, in addition to the arguments given above, mutation is merely assigned a background role in support of recombination

5 MATHEMATICAL FRAMEWORK — HOLLAND'S SCHEMA THEOREM

(This section aims to be a simplified substitute for some more rigorous theoretical treatises found in the literature, and requires only an elementary mathematical background. It unveils the kernel of genetic algorithm mechanics and should therefore be considered a prerequisite to a thorough understanding of genetic algorithms and their successful application to a variety of complex problems. Nevertheless, if skipped there will be no serious loss of continuity in reading this tutorial, provided that the main conclusion is accepted, namely genetic algorithms tend to sample a search space exponentially)

One of Holland's great merits is that he developed a comprehensive mathematical framework, known as the schema theorem [51], which makes explicit selective incremental build-up in genetic algorithms and thereby enables an approximation of the search efficiency He confined himself to a binary representation and the following simple heuristic operators

- Exploitation by simple reproduction (fitness-proportional reproduction)
- Exploration by simple recombination (1X) and simple mutation (M)

Numerous attempts towards mathematical frameworks for genetic algorithms that employ more complex representations and evolutionary search heuristics have been launched (with varying success) and are documented in the literature As the gist of all these frameworks turns out to be the same (Section 6), we too confine ourselves to a simple framework for convenience and clarity. In particular, the framework we outline here uses the same simple heuristic operators that Holland used for his mathematical derivation However, we depart from him by using a more general representation — based on a k-cardinal alphabet instead of a binary alphabet — since this does not change the mathematical derivation, i e it leads to the same final equation

Understanding theoretical issues of genetic algorithms requires a certain attitude We emphasize two points in this First, one should not be concerned with the semantics (meaning) of strings in the population. Semantics are the concern of the implementor He/she chooses a representation and genetic operators to the best of his/her knowledge about the problem that must be solved. Once this is done, the resulting genetic algorithm itself is not (and does not need to be) 'aware' of string semantics — it only needs to look at fitness values in order to know which strings must be reproduced at what rate Likewise, a theory that

describes the genetic search process and explains its properties, uses fitnesses and ignores semantics. The theory does, however, acknowledge the importance of syntax (representation) — a relevant factor in the search efficiency, as was pointed out above. This brings us to the second point: genetic algorithms work most efficiently with problem representations that usually differ from the problem representations that human beings would use to solve problems by hand. This is a characteristic of automated problem solving in general. For instance, a pocket calculator — unlike a human being — uses a binary representation to conduct arithmetic. Accepting this fact is in essence not different from accepting representations by which genetic algorithms process information best.

5.1. Information processing

The first step towards understanding the search behavior of genetic algorithms is acknowledging that the importance of the individual (string) is inferior to the importance of the group (population) and the sub-groups in it. Harsh proof of that is the mortality of the strings, viz. their lifetime is by definition only one generation. Indeed, the evolutionary search heuristics are not really 'interested' in the strings themselves, which they explicitly process, but in fact in the implicit information that they carry. Shortly, we will show how information is passed on to succeeding generations at rates that correspond to its utility. A definition of utility is also given. First, a definition of information is needed, as well as a notation for it.

Information is expressed in the form of a statement, or hypothesis, $\mathscr{H}$, about the true solution. For instance, for a binary representation one may launch the following hypothesis:

$\mathscr{H}$: in the true bitstring, the values of the second, fourth and fifth bit are 1, 0 and 1, respectively.

In more general terms, a particular hypothesis $\mathscr{H}$ indicates some subset, H, of all possible strings that comprise the search space, S, and proclaims that the true string is a member of H, in brief:

$\mathscr{H}$: the true string is a member of H, a subspace of S ($H \subseteq S$).

(The choice of the symbol H derives from 'hyperplane in the search space'.)

Conversely, any subset H drawn from the search space will, by definition, reflect a legal item of information $\mathscr{H}$, regardless of its importance for the search task.

The population, P, is a sample of the search space, i.e. P, like H, is also a subspace of S ($P \subseteq S$). Therefore, P might contain some strings that fulfil $\mathscr{H}$, i.e. might contain some strings that are in H. This cross-section of H and P is denoted as H_P ($H_P = H \cap P$). The subset dependencies of S, P, H and H_P are graphically summarized in Fig. 11.

The size of H is denoted as N_H and is constant for a particular $\mathscr{H}$; we say that H is static. On the other hand, the size of H_P is denoted as N_{H_P} and is variable for a particular $\mathscr{H}$ as H_P depends on the population P, which evolves; we say that H_P is dynamic. Evidently, since H_P is a subset of H, the inequality $N_{H_P} \leqslant N_H$ holds.

In passing, a hypothesis $\mathscr{H}$ is sometimes also said to be shared by the possible strings in static class H, or by the actual strings in dynamic class H_P. Equivalently, one might say that the strings in a class share a certain property. Conversely, the total information stored in a particular string is,

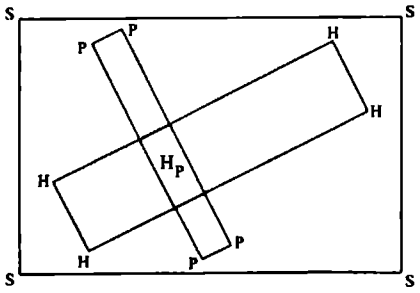

Fig. 11. Schematic illustration of subset dependencies concerning the population (P), a static hyperplane (H) and a dynamic hyperplane (H_P) in the search space (S).

in general, distributed over a number of hypotheses (properties).

The importance of the information $\mathscr{I}$, associated with a static class H, is estimated by the average fitness of the possible strings in H that are actually also in the population P, i.e. by the average fitness of the actual strings that comprise the dynamic class H_P, in brief:

$$\hat{f}_H = \bar{f}_{H_P} \tag{2}$$

where $\hat{f}_H$ is an estimate of the average fitness of the possible strings in H and $\bar{f}_{H_P}$ is the average fitness of the actual strings in H_P. Note that $\hat{f}_H$ is maximal if H contains only the true string.

In contrast to the strings in H_P, which are mortal as their life-span is only one generation, the information $\mathscr{I}$ associated with it might survive owing to fitness-proportional reproduction. However, the size of H_P may grow or decay over successive generations.

The classification of strings according to information shared by them, basically provides a tool by which similarities between strings can be exploited. In the more general context of optimization, this is in fact how genetic algorithms exploit structure on the o-landscape (cf. Section 3.6).

Now that we have detached ourselves from an individual-centered view about genetic algorithm mechanics, the next steps are to discuss a useful notation for classes and then to mathematically describe the evolution of an arbitrary dynamic class. In genetic algorithms parlance, a class is called a schema. In the following we will use the term 'schema' as a shorthand for 'static schema'.

5.2. *Schemata (similarity templates)*

As we pointed out above, we will depart from the binary alphabet. This is to emphasize that schemata can accommodate strings of any kind and not just binary coded strings. In general, a string may be regarded as an l-sized row composed by copying l characters from the k-cardinal alphabet $\mathscr{A}_k = \{a_1, \ldots, a_k\}$ — e.g. {0,1} or {0,1,2,3,4,5,6,7,8,9} in the special case of a binary or decimal alphabet, respectively.

The size of S (the search space) is k^l. Since, according to the earlier definition, a schema H can be any subset in S, it is important to assess how many Hs are contained in S. In general, the number of subsets that can be drawn from J items is 2^J, because for each item there are only two possibilities: it either belongs to a particular subset or not. Hence, the number of Hs contained in S is 2^{k^l}.

Finding a notation by which all 2^{k^l} Hs in S can be described turns out to be cumbersome. Indeed, theoricians simply do with some convenient notation for Hs, realizing that these then no longer cover all possible subsets in S. A convenient (but by no means universal) way — due to Holland [51] — by which a schema can be denoted, is as an l-sized row composed by copying l characters from the $(k+1)$-cardinal alphabet $\mathscr{A}_{k+1} = \{a_1, \ldots, a_k, *\}$ — e.g. the ternary alphabet {0,1,*} in the special case that $\mathscr{A}_k$ is binary. The character * by which $\mathscr{A}_k$ is in fact extended to obtain $\mathscr{A}_{k+1}$, means 'any character in $\mathscr{A}_k$ matches here'.

Example. For $k = 2$ and $l = 5$ the schema $H = \{*1*01\}$ represents the following class with 4 bitstrings:

```
{0 1 0 0 1,
 0 1 1 0 1,
 1 1 0 0 1,
 1 1 1 0 1}
```

As the meta-character * is merely a notational device that allows the formal description of similarities among strings, it is important to realize that it is never explicitly processed by a genetic algorithm. Schemata based on a meta-character are called Holland schemata [53–55], in order to emphasize that they are just one of several kinds that may be defined. For meaningful values of k and l ($k \geqslant 2$ and $l \geqslant 1$) it is easily verified that the number of Holland schemata contained in the search space S is always less than the number of all possible subsets in S, as argued before: $(k+1)^l < 2^{k^l}$. Despite their limitation, Holland schemata are useful because they give access to a comprehensive theory — the schema theorem — from which important general conclusions can be drawn.

In the following, the term 'schema' will be used in either the general sense or in a special-

ized sense (e.g. the Holland sense). The exact sense that applies should be clear from the context.

Holland schemata have two elementary properties, which are important for the derivation of the schema theorem. The first is called schema order, $\mathscr{O}$, which is the number of fixed (non-*) positions — e.g. 3 in the example above. The second property is called the defining length, $\mathscr{D}$, of a schema, which is the difference between the index of the last fixed position and the index of the first fixed position — e.g. $5 - 2 = 3$ in the example above.

At this juncture, one can make the following important counting observations (which are easily verified, see comment between parentheses):

- The search space S comprises k^l strings — e.g. 2^l strings for $k = 2$ (because at each of the l positions: either a_1 or ... or a_k).
- The search space S comprises $(k+1)^l$ schemata — e.g. 3^l schemata for $k = 2$ (because at each of the l positions: either a_1 or ... or a_k or *).
- A particular schema H of order $\mathscr{O}$ comprises $N_H = k^{l-\mathscr{O}}$ strings — e.g. $2^{l-\mathscr{O}}$ for $k = 2$ (because at each of the $l - \mathscr{O}$ *-positions: either a_1 or ... or a_k).
- A particular string comprises 2^l schemata — for any k (because at each of the l positions: either a_i or *, where i = position index).

From the first and the second counting observations it can be seen that the ratio of schemata to strings in a search space is highest for the lowest relevant k, i.e. $k = 2$. This fact suggests that the maximum feasible wealth of information is contained in a binary population, and has often led to the claim that the most efficient search can be attained for a binary representation [2]. However, this assertion ignores the additional role that search heuristics play in the search efficiency (principle of optimal method). Indeed, practice has shown that non-binary representations are sometimes eligible (see Part 2 [37]).

The last counting observation allows us to calculate a range for the number of schemata contained in a population P comprising N_P strings: when all strings in P are identical, then the number of schemata contained in P is 2^l (minimum), whereas when there is maximal diversity in P, then the number of schemata contained in P is $N_P \cdot 2^l$ (maximum). Note that numbers within this range approach the order of the size of the search space, and thus give us an impression of the vast amount of information typically being processed by a genetic algorithm during a particular generation. However, in order to really understand the evolution of information we must analyze the growth and decay of individual dynamic schemata, which leads us to the schema theorem. In brief, that schema theorem comprises an expression for the expected minimal growth rate of a particular dynamic schema H_P. Simply stated, this expression is a factorization of two effects: an effect due to exploitation and an effect due to exploration, as described in Section 4.2.

5.3. Exploitation effect

At time step (generation) t, consider a population $P(t)$, a schema H, and a dynamic schema $H_P(t)$ which is a subset of H and contains $N_{H_P(t)}$ actual strings from $P(t)$. Let $\bar{f}_{P(t)}$ be the average fitness of $P(t)$. Let $\bar{f}_{H_P(t)}$ be the average fitness of $H_P(t)$, which, by Eqn. 2, equals $\hat{f}_H$: the estimated fitness of schema H.

After fitness-proportional reproduction, the next generation $t+1$ results in population $P(t+1)$, and a dynamic schema $H_P(t+1)$ which is a 'grown (or decayed) version' of $H_P(t)$, i.e. which is a new subset of the same H. It can easily be shown [2,51] that $N_{H_P(t+1)}$ — the expected number of actual strings that $H_P(t+1)$ contains from $P(t+1)$ — is given by:

$$N_{H_P(t+1)} = N_{H_P(t)} \cdot \frac{\hat{f}_H}{\bar{f}_{P(t)}} \tag{3}$$

This means that a particular dynamic schema grows as the ratio of the estimated average fitness of the schema H to the average fitness of the population. That is, schemata with fitness values above the population average will receive an increasing number of instances (actual strings) in the next generation, while schemata with fitness values below the population average will receive a decreasing number of instances. Importantly,

this expected behavior is carried out with every schema contained in the population — in parallel and with no special bookkeeping or special memory other than the population itself! It should be noted, however, that there is a statistical error in the estimate $\hat{f}_H$, called the sampling error. This may lead the genetic algorithm astray, especially if the population size N_P is relatively small.

For the moment, assume that $\hat{f}_H/\bar{f}_{P(t)}$ is equal to a constant, c, i.e. time-independent. Eqn. 3 then simplifies to:

$$N_{H_P(t+\tau)} = N_{H_P(t)} \cdot c^{\tau} \tag{4}$$

Note the exponential dependency on the elapsed number of generations, τ. Slightly relaxing the assumption on which this Eqn. 4 is based, it is still reasonable to assume [2] that fitness-proportional reproduction will, by approximation, allocate exponentially increasing (or decreasing) numbers of instances to above- (or below-)average schemata. Such near-exponential search behavior is an argument in support of the widespread conviction that genetic algorithms tend to solve complex large-scale problems in polynomial time (Section 3.8.2).

5.4. Exploration effect

As argued before, exploration is required in order to counterbalance depletion of the population caused by exploitation (fitness-proportional reproduction). Exploration comes down to variation, imposed on the strings by recombination and mutation, and leads to the best search efficiency if the principle of maximal preservation is observed. Preservation (or loss of information, respectively) can now be quantified in terms of schemata as follows.

Recall that any string of length l comprises 2^l schemata. Therefore, when such a string recombines or mutates, it switches from one set of 2^l contained schemata to another. The original and the new schemata set will, in general, not mutually overlap completely, i.e. some of the original schemata have 'vanished'. Thus, taking all strings of the population into account, the total impact of recombination and mutation across the population might be that a particular schema vanishes entirely from the population. Such a schema is said to be disrupted, i.e. the information it represents is lost from the population.

Using the earlier defined schema properties $\mathscr{O}$ (order) and $\mathscr{D}$ (defining length), the disruptive effects of recombination and mutation as originally chosen by Holland, can be easily estimated. In particular, let the mutation (**M**) probability be p_m (the chance by which a local change in the encoding occurs anywhere in the population) and the recombination (**1X**) probability be p_r (the chance by which randomly paired strings will actually exchange segmental fractions of their code). It follows [2,51] that a particular dynamic schema, H_P, will now receive an expected number of instances in the next generation under reproduction, recombination, and mutation as approximated by the following inequality — known as the *schema theorem*:

$$N_{H_P(t+1)} \geq N_{H_P(t)} \cdot \frac{\hat{f}_H}{\bar{f}_{P(t)}} \cdot \left[1 - p_r \cdot \frac{\mathscr{D}_H}{l-1} - \mathscr{O}_H \cdot p_m\right] \tag{5}$$

The schema theorem corresponds with Eqn. 3 extended by a loss (disruption) factor. Since mutation is usually assigned a background role, it may be further approximated by:

$$N_{H_P(t+1)} \geq N_{H_P(t)} \cdot \frac{\hat{f}_H}{\bar{f}_{P(t)}} \cdot \left[1 - p_r \cdot \frac{\mathscr{D}_H}{l-1}\right] \tag{6}$$

We emphasize that the loss factor in this equation does not apply generally, but only for genetic algorithms that employ **1X** recombination. In that case, loss of important information is minimized by minimizing $\mathscr{D}_H$. How this can be done leads to the concept of building blocks, discussed in the next section.

5.5. Building blocks

From Eqn. 6, near-exponential search behavior may again be expected provided schema disruption is not too severe. More precisely [2,51]:

With **1X** as the recombination operator, schemata with high estimated fitness ($\hat{f}_H$) and

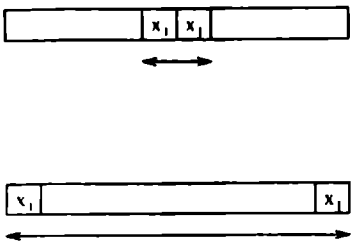

Fig 12 Effect of placement of problem parameters, on the length of the segment wherein 1X recombination acts disruptively for correlated problem parameters x_i and x_j.

short defining length ($\mathscr{D}_H$) are encouraged generation to generation by giving near-exponentially increasing instances to the observed best. Such schemata are called building blocks.

By the growth of building blocks we will understand the phenomenon whereby building blocks increasingly clump together within the strings. These clumps are often called 'larger building blocks' and are often viewed as larger partial solutions. However, strictly speaking, large building blocks do not exist, because a building block is short by definition. The following example elucidates, using 1X recombination, the dramatic impact that representation can have on the formation and growth of building blocks.

Consider a problem that is not free of epistasis (Section 3.8.3), i.e. at least two problem parameters, say x_i and x_j, are correlated in the sense that high fitness values can only be attained when x_i and x_j adopt certain values simultaneously. Next, consider two binary representations: in the first the bitfields for x_i and x_j are placed adjacent to each other on the bitstrings, in the second they are placed at the ends of the bitstrings (Fig. 12). Since any combination of values for x_i and x_j that incidentally leads to a high fitness for a particular bitstring is more likely to be destroyed by 1X in the second representation than in the first, the difference in the ease by which building blocks can form and grow is evident. Accordingly, 1X is said to exhibit positional bias [56].

Positional bias of the traditional 1X recombination is not necessarily a disadvantage in practice. In fact, 1X may be a highly efficient recombination for problems with not too many correlated problem parameters. The implementor will encode these parameters adjacently on the strings. However, the positional bias of 1X is cumbersome in problems in which many problem parameters are correlated. Other recombination operators, viz. operators free of positional bias, are then desirable. Such an operator is discussed in Part 2 [37].

In passing, we note that genetic algorithms which employ 1X recombination have the educational advantage that building blocks grow segment-wise: they provide an intuitively appealing model of the principle of selective incremental build-up. For more complex recombination operators, the notion of building blocks remains valid, only the latter can then no longer be pictured in the intuitively appealing segmental way. All that matters is the concept that building blocks are the 'good information' fractions of each of the two parent strings that are to be combined in child strings. (Of course, one can also get together the 'bad information', but the ensuing selection pressure filters that out automatically.)

The growth of building blocks is such an important distinguishing feature of genetic algorithms, that there is a special term for what we have in this tutorial called the principle of selec-

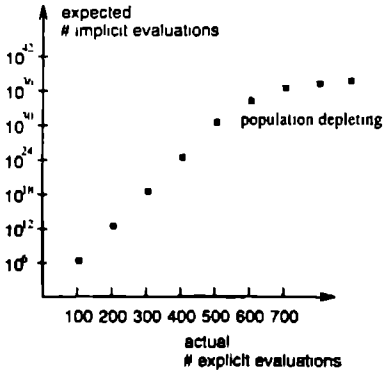

Fig. 13. Implicit parallelism for population size 100. The number of implicit evaluations are approximate.

tive incremental build-up (the principle of maximal preservation, Section 4.2), namely: the building block principle.

5.6. *Implicit parallelism*

Holland derived the following estimate for the total number of schemata that are effectively processed in an N_P-sized population [51]: on the order of N_P^3 schemata are implicitly processed as N_P strings are explicitly processed per generation, provided the population is sufficiently diverse.

This property was considered so important and apparently unique to genetic algorithms that Holland gave it a special name: implicit parallelism. Its implications are graphically illustrated in Fig. 13.

6. GENERALIZING THE SCHEMA THEOREM

In several ways, the implications of Holland's schema theorem go beyond the assumptions on which it is based. First, an important point to understand is that there is no a priori reason why information processed by a genetic algorithm should be represented by schemata based on a notation with a meta-character (*) [53–55]. We argued before that such a notation constrains the number of subsets that can be drawn from the search space. (Recall that any subset in the search space is a legal schema, by definition.) Indeed, a genetic algorithm may well process non-Holland schemata of any kind — i.e. information that we perhaps do not even know how to explicitly represent. All that matters is that schemata — hyperplanes — of some kind can be processed profitably, given the factors that determine which information will be processed, namely: a representation and evolutionary search heuristics. A genetic algorithm, then, will at any time automatically 'navigate' the most rewarding hyperplanes. On the other hand, it must be emphasized that as schemata basically represent the direction of genetic search, it is important that, at some stage, they contain the object of search: the true solution. When this is not the case, a genetic algorithm is said to be deceived [2], and another combination of representation and evolutionary heuristics may then lead to better performance.

Second, as argued before, the building block principle is plausible by itself and in that sense not confined by the assumptions on which the schema theorem is based. In particular, the disruption term in the schema theorem merely proposes mathematically how the building block principle should be realized in the special case of **1X** recombination (and **M** mutation). Under different circumstances, the building block principle should be realized differently. However, for many kinds of search tasks it seems difficult to define the kind of schemata that would be best to process, let alone to develop a mathematical framework in terms of them. Fortunately for genetic algorithm practitioners, it turns out that a tailored mathematical theory is not a sine qua non for good performance. Practitioners typically apply the building block principle intuitively, starting from plausible assumptions of what partial solutions look like and how they should be combined to create better (partial) solutions. More detail about making such educated guesses is provided in Part 2 [37].

Third, near-exponential search behavior is primarily a consequence of only one of the assumptions on which the schema theorem is based, namely fitness-proportional reproduction. When the other assumptions — **1X** recombination and **M** mutation — are relaxed, this behavior is still to be expected (assuming low disruption).

More popularly, the generalization of the schema theorem may be summarized as follows. Genetic algorithms possess an 'evolutionary memory' based on the processing of similarities between strings. These similarities are implicitly evaluated as the strings are explicitly evaluated. That is, explicit evaluation of a string implies, in effect, partial evaluation of other strings — either actually in the population or not — which are similar to some extent. In practice, the impact of this is such that highly performing regions in the search space can be quickly identified. Sampling in the search space is then increasingly focused on these regions at the expense of the low performing regions.

7 PROPERTIES OF GENETIC ALGORITHMS

Although the near-exponential search efficiency is a salient feature of genetic algorithms, their overall judgment should include other properties as well

In general, the versatility of — or the configurational flexibility in — genetic algorithm methodology is attractive, although more so to the experienced genetic algorithm implementor than to the novice Configuration of a genetic algorithm requires the selection of a flowchart and its constituent modules (e g the recombination procedure), as well as the setting of the numerical parameters that control the action of the modules (e g the probability, p_r, of recombination) In general, the configurational space of genetic algorithms is huge and there is no standard methodology available to find an optimal working point in this space Accordingly, only with a reasonable understanding of genetic algorithm mechanics can one usually make an educated guess at a productive working point However, it turns out that even if a non-optimal working point is chosen, the global solution can often still be found, ultimately, although more time steps are on the average required Therefore, genetic algorithms are configurationally fairly robust In passing, we note that large configurational spaces are a typical property of natural computational methods in general (Section 8)

We argued before that the versatility of genetic algorithms is related to the fact that they do not make strong assumptions about the shape of the fitness landscape (o-landscape) For the same reason, genetic algorithms are also robust in another sense than the aforementioned configurational sense, viz they tend to approach the global solution quite independently from the starting population in the search space Thus, genetic algorithms are characterized by a good search accuracy

On the other hand, however, genetic algorithms appear to exhibit a poor search precision — a considerable spread in estimated solutions about the global optimum upon replicating runs This spread is to be ascribed to the stochastic search heuristics used by genetic algorithms, and it may be indicated as method noise

Opposite properties often apply to traditional local optimization techniques a good search precision and a poor search accuracy Therefore, better performance might be attained in a sequential combination, where the genetic algorithm generates a 'best guess' that serves as a starting point for subsequent refinement by the local technique Such, and other hybrid techniques, wherein separate methods mutually supplement each other for enhanced overall performance, are outlined in Part 2 [37]

8 GENETIC ALGORITHMS IN NATURAL COMPUTATIONAL PERSPECTIVE

It appears that genetic algorithms fit within a broader methodological context — a recently increasingly acknowledged paradigm, called natural computation Natural computation refers to all problem solving methods that are founded upon a natural system or process As the resemblance between such a system or process and the method derived from it is very loose, in general, one practically important property that natural computational methods share is that there is virtually no limit to the number of method configurations that might be useful in practice

Tentatively, two main branches may be distinguished within natural computation — (1) optimization and (2) simulation and modeling — which both can be further subdivided

Natural computation

- Optimization
  - Simulated annealing
  - Simulated evolution
    - Evolution strategies
    - Evolutionary programming
    - *Genetic algorithms*
- Simulation and modeling
  - Classifier systems
  - Artificial immune networks
  - Artificial neural networks

- Artificial chemical reaction systems
- Artificial life (related to cellular automata and fractals)

8.1. Optimization

Both simulated annealing and simulated evolution methods are based upon stochastic search heuristics wherein an explorative and an exploitative mechanism cooperate. Under certain mathematically well-defined circumstances it can sometimes be proved that exploration excludes the possibility that the global solution will never be found, i.e. ensures that the global optimum must ultimately be found. (Whether this is within practical time constraints is an open question, though.) Exploitation, on the other hand, can for some problems be shown to cause convergence in polynomial time (Section 3.8.2).

Problem solving with simulated annealing mimicks the heating and then slow cooling of a solid to remove strain and crystal imperfections [57–59]. In this method, point-to-point steps are taken within the search space that are increasingly local as the 'temperature' becomes lower according to an imposed 'cooling' schedule. Hitherto, only a few comparative studies involving simulated annealing methods and simulated evolution methods have been conducted. For a combined treatise of simulated annealing and genetic algorithms, the reader is referred to ref. [60]. Some analytical-chemical applications of simulated annealing are described and referenced in ref. 61.

As pointed out before, the correspondence between simulated evolution methods is that they use a population of string-represented candidate solutions as an inherently parallel memory device to guide the search, and that they periodically update this population under selection pressure, followed by semi-random variation. Evolution strategies [62–65] originated and were developed in Germany from the early nineteen-sixties. Along an independent thread, evolutionary programming [66–69] and genetic algorithms originated and developed in the United States of America.

While genetic algorithms and evolutionary programming allow any type of encoding (binary, decimal, real, etc.), evolution strategies insist on a real encoding of the problem parameters. The exploratory mechanism of evolution strategies lies emphasis on mutation rather than on recombination. Mutation is conducted by incrementing the value of the problem parameters with a Gaussian-distributed step that has zero mean and a variance that is adaptively adjusted during the search process. Recently, evolution strategies have been extended with representational mappings in order to encompass combinatorial optimization as well [70].

Like evolution strategies, evolutionary programming is sceptical about recombination as a constructive assembly mechanism that adds to the search efficiency. Even more so, advocates of evolutionary programming condemn recombination as essentially an incorporation of a greedy algorithm into the simulated evolution process, which may therefore limit robustness. Mutation, on the other hand, is praised as it is simple and therefore does not bias the search.

In sharp contrast to evolutionary programming and evolution strategies, genetic algorithms are 'sexual', i.e. they view recombination as a constructive assembly mechanism which may combine building blocks — fractions of useful infor-

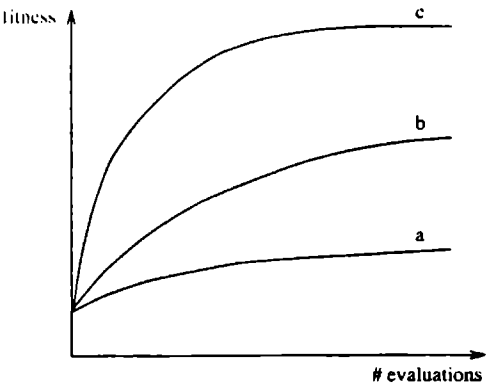

Fig. 14. Comparison of stochastic search heuristics for a problem for which a constructive recombination operator exists. (a) Only mutation (at high rate — basically leads to Monte Carlo search). (b) Mutation (at high rate) and selective reproduction. (c) Recombination (assisted by mutation at low rate) and selective reproduction.

mation — partial solutions — into larger building blocks in a serendipitous way (cf. Fig. 10). Therefore, with a properly working recombination operator, one would expect genetic algorithms to accelerate the optimization process in comparison to 'asexual' simulated evolution methods (evolution strategies and evolutionary programming) and Monte Carlo search (Fig. 14). Whether genetic algorithms can indeed reach a higher search efficiency for a particular problem remains an issue of further comparative studies. In the meantime it should be noted that for many problems it is not an easy task to design a recombination operator that can automatically recognize the proper kind of building blocks in string-represented candidate solutions and combine these in a meaningful way into larger building blocks such that new strings with higher fitness values can be created.

8.2. Simulation and modeling

All methods in the learning branch of the natural computation framework have the common property that they attempt to simulate or model a system or process. There is a training stage — which is basically an optimization procedure wherein the objective function is based on a criterion that is called a learning rule — and a prediction stage.

We only shed some light on classifier systems, as these incorporate a genetic algorithm. Classifier systems comprise a class of learning systems, developed by Holland in the early eighties [43,71–76]. In brief, a classifier system mimicks the behavior of a dynamical system, i.e. a process. An often cited example is Goldberg's classifier system which optimizes the operation of an artificial gas pipeline system [2,77–79]. He focused on a ten-compressor, ten-pipe, steady-state, serial pipeline problem. The aim was to minimize the power consumption, subject to maximum and minimum pressure and to pressure ratio constraints.

In general, a classifier system consists of an input unit which reads signals from a dynamic system, and an output unit which emits signals that reflect the mimicked system behavior. As this behavior is compared with the actual system's behavior, the learning is guided. Between the input unit and the output unit stands a population that consists of so-called classifiers — simple production rules with a ternary-coded condition part and a binary-coded action part. Classifiers are sometimes compared with lymphocytes in an immune system [80,81]. Immune systems have proved to be very sophisticated learning systems for biological pattern recognition and defensive task execution in both stationary and dynamic environments, and their simulation by use of classifier systems therefore seems to be potential. In general, a classifier system comprises two algorithms:

- A bucket-brigade algorithm for rule-enforcement.
  Evaluation of the system's behavior leads to an inflow of fitness into the population. This fitness is distributed over the classifiers according to a mechanism known as a bucket brigade algorithm. It treats the population as a complex microeconomy by controlling 'trade' interactions (fitness transactions) between classifiers. A bucket brigade algorithm comprises the so-called 'minor cycle' of a classifier system (short time-scale), as classifiers may be strengthened or weakened at every iteration.
- A genetic algorithm for search.
  After a predefined number of cycles of the bucket-brigade algorithm, a genetic algorithm is activated as the so-called 'major cycle' of a classifier system (long time-scale). This background action merely serves to guarantee that every corner in the procedural search space has a non-zero chance to be sampled.

Although the ideas behind classifier systems are appealing, they are only about one decade old and actually still plagued with some difficulties as well — e.g. the tuning of the major cycle to the nested minor cycle. Unlike genetic algorithms for optimization (which have been in existence for over two decades now), classifier systems have not yet matured to the point where they are used to solve real-world problems, and they probably will not be in the short term, either [82]. Another

serious problem of classifier systems is the competition that they suffer from the more popular and widely established artificial neural networks It is premature to tell whether classifier systems stand a reasonable chance to ever break through in the future.

For more details on artificial immune networks see, e g., refs 83–85, on artificial neural networks see, e.g., refs. 86,87; on artificial chemical reaction systems see, e g , refs 88–90, and on artificial life see, e g., refs. 91–94

8 3. Natural computation

There is an important unifying notion behind natural computation methods [84] The information about the natural system or process that is mimicked by such a method is stored in an ensemble of identical units — 'neurons' in artificial neural networks, 'strings' in simulated evolution methods and classifier systems, 'chemicals' ('adducts' and 'products') in artificial chemical reaction systems, 'lymphocytes' in artificial immune networks Each unit in the ensemble possesses simple information processing capabilities owing to a small number of properties The properties comprise a means for mutual interactions between the units These interactions are controlled by a finite set of comparatively simple computation rules that are, effectively, applied massively in parallel in each time step Computational power is hidden in the complexity of the interactions between the units These interactions are often called connectionist if between 'neurons', selectionist if between 'strings', reactionist if between 'chemicals', and immunist if between 'lymphocytes'. Due to the nonlinear impact of the interactions, properties at the level of ensemble units will collectively transform into novel, or emergent, properties at the global level, wherein computational power is hidden. (Emergent properties in natural systems are familiar, e g a phase transition is a property that can only arise when atoms or molecules form an ensemble.) The emergence of structure in the system is a spontaneous complex-dynamic process, i.e. essentially without any central supervising agent for the manipulation of data representations. The system is therefore said to be capable of self-organization (learning). When there are strong nonlinearities in the system, then an aspect of unpredictability, or chaos, may arise, along with the process of self-organization, which leads to yet other computational frontiers

Although it is widely acknowledged that the complex-dynamic collective behavior in self-organizing systems contains tremendous potential, in principle, for complex problem solving, the exploitation and manipulation of this power does not turn out to be very straightforward and requires at least a good intuitive understanding of the mechanics of such systems Nevertheless, further maturation and consolidation of natural computational sciences is to be expected in the coming years, and these developments are most likely to make inroads into computational chemistry as well

9 SOFTWARE

Building a genetic algorithm requires the design of a flowchart and its constituent modules — the genetic routines for reproduction, recombination and mutation, and other routines. Numerous techniques for reproduction, recombination and mutation are known from the literature As these techniques are usually not restricted to one problem only, they are to a certain degree domain-independent. Ideally, therefore, genetic algorithm software comes as a package, or library, that contains a number of such 'domain-independent' routines that need not be 'aware' of the meaning of the string-represented candidate solutions that they operate on.

The library serves as a 'kernel', or genetic engine, that may be employed to 'drive' numerous applications. Evidently, using the same genetic engine for different applications facilitates implementation and thus results in very time-efficient development of applications. The precise selection of the library routines in building an application ultimately depends on the nature of the problem that needs to be solved. Another argument in favor of library usage is flexibility· many kinds of flowcharts can be assembled by

combining the appropriate routines in the library, while new 'domain-independent' routines can always be added to the library in order to extend the scope of problems that it can potentially handle. However, despite the fact that the library may be applicable to a very wide spectrum of problems, it cannot possibly anticipate on all problems for which a genetic algorithm would be a profitable choice: an arbitrary problem might always require an unprecedented representation and/or search heuristic.

Besides 'domain-independent' routines from the library, a genetic algorithm requires domain-dependent modules such as the definition of a problem representation, an evaluation criterion (objective function), a termination criterion, and I/O facilities for control and for monitoring of performance. That is, the definition of the domain always requires the specification of procedural data besides factual input data, i.e. always requires programming. Accordingly, general-purpose genetic algorithm software is not suitable for the end-user, but rather so (e.g. as a library) for the programmer who creates an end-user application on a consultant–client basis.

The need for programming in creating a genetic algorithm for each specific problem should, however, not be viewed as a disadvantage — it is merely a consequence of the versatility of genetic algorithm methodology. Once an end-user application has been developed for a particular problem, it can be used for numerous versions of that problem — versions that differ only in terms of the input data used. For instance, a genetic algorithm for curve fitting treats any new input spectrum as a novel problem, and may therefore stay in use on a routine basis for a long time. In this way, an application may ultimately pay back the prior investment in its development.

In our laboratory, a 'general-purpose' genetic algorithms library has been developed over the past few years, called GATES: Genetic Algorithm Toolbox for Evolutionary Search [96], primarily for internal use. Although recently other genetic algorithms software has been developed — e.g. GENESIS (J.J. Grefenstette), GENITOR [97], OOGA [98], GAucsd (N.N. Schraudolph), GA Workbench (M. Hughes), Evolution Machine (H.-M. Voigt), GAGA (J. Crowcroft), Splicer [99], and more — we have chosen to create our own software in order not to be limited by scope and not to be dependent on software support.

The structure of GATES is such that the routines are layered in hierarchies. At the lowest level there are tedious primitive routines, e.g. for bit manipulations related to bitfield decoding and routines for pseudo-random number generation. At an intermediate level there are numerous reproductions schemes, recombination operators and mutation operators that can be applied to a wide range of problems in problem classes NUM, SEQ and SUB (Section 2.1). At this level, the implementor has great freedom in giving shape to the configuration of his/her flowchart. At the highest level there are prefabricated flowcharts for NUM, SEQ and SUB problems, along with control setting files to parameterize the genetic routines. By choosing a prefabricated flowchart, the implementor deliberately limits his/her configurational freedom for the sake of minimal implementation effort: only an evaluation function needs to be programmed and linked to the library in order to obtain the end-user application. Auxiliary facilities have recently been added to GATES, among which we mention procedures for empirical complexity analysis of the o-landscapes (Section 3.8.1) as an aid in, inter alia, configuration and control setting.

For those who are interested in a practical encounter with genetic algorithms, a demonstration program which visualizes the mechanism underlying genetic algorithms — VGA (visual genetic algorithm) — is available on request. VGA is built upon GATES and was written by W.J. Melssen and C.B. Lucasius. Due to the incorporated graphics, VGA is necessarily limited to a single type of platform — a SUN workstation in an X-windows environment.

10. CONCLUSIONS AND OUTLOOK

The mechanics and characteristics of genetic algorithms have been elucidated and placed in their broader context. It follows that genetic algorithms are efficient, efficacious and widely appli-

cable to real-world problems. Many computational-chemical problems too, by their complex and large-scale nature, seem highly suitable for an approach by genetic algorithms. Applications are expected to increase over the coming years as the methodology is increasingly recognized in many fields of science.

ACKNOWLEDGMENTS

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007. We wish to thank the reviewers for their critical comments and useful suggestions.

REFERENCES

1 J.J. Grefenstette (Editor), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
2 D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
3 R. Dawkins (Editor), *The Blind Watchmaker*, Penguin Books, London, 1986.
4 C. Darwin (Editor), *The Origin of Species by Means of Natural Selection: the Preservation of Favoured Races in the Struggle for Life*, John Murray, London, 1859 (first edition text, edited with an introduction by J.W. Burrow, published by Pinguin Books, 1968).
5 L. Davis (Editor), *Handbook of Genetic Algorithms, Part I: A Genetic Algorithms Tutorial*, Van Nostrand Reinhold, New York, NY, 1991, pp. 1–101.
6 J.H Holland, Genetic algorithms, *Scientific American*, July (1992) 44–50.
7 Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Artificial Intelligence Series, Springer-Verlag, Berlin, 1992.
8 G. Kateman, Evolution in chemometrics, *The Analyst*, 115 (1990) 487–493.
9 M.J.J. Blommers, C.B. Lucasius, G. Kateman and R. Kaptein, Conformational analysis of a dinucleotide photodimer with the aid of the genetic algorithm, *Biopolymers*, 32 (1992) 45–52.
10 G.C. Levy, S. Wang, P. Kumar and P. Borer, Multidimensional nuclear magnetic resonance spectroscopy and modeling of complex molecular structures: A challenge to today's computer methods, *Spectroscopy International*, 3 (4) (1991) 22.
11 C.B. Lucasius and G. Kateman, Application of genetic algorithms in chemometrics, in J.D. Schaffer (Editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989, p. 170–176.
12 C.B. Lucasius, M.J.J. Blommers, L.M.C. Buydens and G. Kateman, A genetic algorithm for conformational analysis of DNA, in L. Davis (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991, Ch. 18, pp. 251–281.
13 C.B. Lucasius, M.J.J. Blommers, S. Werten, A.H.J.M. van Aert and G. Kateman, Conformational analysis of DNA using genetic algorithms, in H.-P. Schwefel and R. Männer (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1991, pp. 90–97.
14 M. Bos and H.T. Weber, Comparison of the training of neural networks for quantitative X-ray fluorescence spectrometry by a genetic algorithm and backward error propagation, *Analytica Chimica Acta*, 247 (1991) 97–105.
15 C.B. Lucasius and G. Kateman, Genetic algorithms for large-scale optimization in chemometrics. An application, *Trends in Analytical Chemistry*, 10 (1991) 254–261.
16 H.M. Cartwright and G.F. Mott, Looking around: using clues from the data space to guide genetic algorithm searches, in R.K. Belew and L.B. Booker (Editors), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 108–114.
17 E. Fontain, The problem of atom-to-atom mapping. An application of genetic algorithms, *Analytica Chimica Acta*, 265 (1992) 227–232.
18 A.P. de Weijer, C.B. Lucasius, L.M C. Buydens, G. Kateman and H.M. Heuvel, Using genetic algorithms for an artificial neural network inversion, *Chemometrics and Intelligent Laboratory Systems*, in press.
19 D. Wienke, C.B. Lucasius and G. Kateman, Multicriteria target vector optimization of analytical procedures using a genetic algorithm. Part 1: Theory, numerical simulations and application to atomic emission spectroscopy, *Analytica Chimica Acta*, 265 (1992) 211–225.
20 S. Schulze-Kremer, Genetic algorithms for protein tertiary structure prediction, *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, Elsevier, Amsterdam, 1992, pp. 391–400.
21 R. Wehrens, C.B. Lucasius, L. Buydens and G. Kateman, HIPS, a hybrid self-adapting expert system for NMR spectrum interpretation using genetic algorithms, *Analytica Chimica Acta*, in press.
22 C.B. Lucasius and G. Kateman, Towards solving subset selection problems with the aid of the genetic algorithm, in R. Männer and B. Manderick (Editors), *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, Elsevier, Amsterdam, 1992, pp. 239–247.

23 R Wehrens, C B Lucasius, L Buydens and G Kateman, Sequential assignment of 2D NMR spectra of proteins using genetic algorithms, *Journal of Chemical Information and Computer Sciences*, in press

24 D Wienke, C B Lucasius, M Ehrlich and G Kateman, Multicriteria target vector optimization of analytical procedures using a genetic algorithm Part 2 Optimization of the photometric calibration curve of dry reagent strips for qualitative clinical analysis, *Analytica Chimica Acta*, in press

25 T H Li, C B Lucasius and G Kateman, Optimization of calibration data with the dynamic genetic algorithm, *Analytica Chimica Acta*, 268 (1992) 123–134

26 A P de Weijer, C B Lucasius, L M C Buydens, G Kateman, H M Heuvel and H Mannee, A new approach to curve fitting using natural computation, *Analytica Chimica Acta*, submitted for publication

27 C B Lucasius, A P de Weijer, L Buydens and G Kateman, CFIT a genetic algorithm for survival of the fitting, *Chemometrics and Intelligent Laboratory Systems*, in press

28 R M Lopes Marques, P J Schoenmakers, C B Lucasius and G Kateman, Modelling chromatographic behaviour as a function of pH and solvent composition in RPLC, *Chromatographia*, in press

29 M Eigen, The physics of molecular evolution, *Chemica Scripta*, 26B (1986) 13

30 M Eigen, New concepts for dealing with the evolution of nucleic acids, in *Cold Spring Harbor Symposia on Quantitative Biology*, Vol LII, Cold Spring Harbor Laboratory Press Cold Spring Harbor, NY, 1987, p 307

31 M Eigen *Steps toward Life A Perspective on Evolution*, Oxford University Press, Oxford, 1992 (translation)

32 P Schuster, The physical basis of molecular evolution, *Chemica Scripta*, 26B (1986) 27

33 Q Wang, Optimization by simulating molecular evolution, *Biological Cybernetics*, 57 (1987) 95

34 X-L Wu and R Freeman, Darwin's ideas applied to magnetic resonance The marriage broker, *Journal of Magnetic Resonance*, 85 (1989) 414

35 X L Wu, P Xu and R Freeman, Delayed focus pulses for magnetic resonance imaging An evolutionary approach, *Magnetic Resonance in Medicine*, 20 (1991) 165

36 P Guntert and K. Wuthrich, Improved efficiency of protein structure calculations from NMR data using the program DIANA with redundant dihedral angle constraints, *Journal of Biomolecular NMR*, 1 (1991) 447

37 C B Lucasius and G Kateman, Understanding and using genetic algorithms Part 2 Representation, configuration and hybridization, *Chemometrics and Intelligent Laboratory Systems*, submitted

38 P R Adby and M A H Dempster, *Introduction to Optimization Methods*, Chapman and Hall, London, 1974

39 M R Garey and D S Johnson, *Computers and Intractability A Guide to the Theory of NP Completeness*, W H Freeman, San Francisco, CA, 1979

40 P E Gill, W Murray and M H Wright, *Practical Optimization*, Academic Press, London, 1981

41 A Torn and A Žilinskas, *Global Optimization*, Springer-Verlag, Berlin, 1989

42 D E Goldberg, The theory of virtual alphabets, in H-P Schwefel and R Manner (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1991, pp 13–22

43 J H Holland, Escaping brittleness the possibilities of general-purpose learning algorithms applied to parallel rule-based systems, in Caianello et al (Editors), *Machine Learning II*, Morgan Kaufmann, San Mateo, CA, 1986, p 593

44 S Kauffman, Towards a general theory of adaptive walks on rugged landscapes, *Journal of Theoretical Biology*, 128 (1987) 11

45 S Kauffman, Adaptation on rugged fitness landscapes in L. Stein (Editor) *Lectures in the Sciences of Complexity*, Addison-Wesley, CA, 1989, p 527

46 E Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference, *Biological Cybernetics*, 63 (1990) 325

47 M Lipsitch, Adaptation on rugged landscapes generated by iterated local interactions of neighboring genes, in R K Belew and L B Booker (Editors), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp 128–135

48 B Manderick, M de Weger, and P Spiessens, Genetic algorithms and the structure of the fitness landscape, in R K Belew and L B Booker (Editors) *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991 pp 143–150

49 Y Davidor, *Genetic Algorithms and Robotics A Heuristic Strategy for Optimization*, World Scientific, Singapore, 1991

50 Y Davidor, Epistasis variance a viewpoint on GA hard ness, in G J E Rawlins (Editor), *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp 23–35

51 J H Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975 Revised MIT Press, Cambridge, MA, 1992

52 J J Grefenstette, Incorporating problem specific knowledge into genetic algorithms, in L. Davis (Editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, Los Altos, CA, 1987, Ch 4, pp 42–59

53 G E Liepins and M D Vose, Representational issues in genetic optimization, *Journal of Experimental and Theoretical Artificial Intelligence*, 2 (1990) 101

54 M D Vose and G E Liepins, Schema disruption, in R K Belew and L B Booker (Editors), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp 237–242

55 M D Vose, Generalizing the notion of schema in genetic algorithms, *Artificial Intelligence*, 50 (1991) 385

56 L J Eshelman, R A Caruana and J D Schaffer, Biases in the crossover landscape, in J D Schaffer (Editor), *Proceedings of the Third International Conference on Genetic*

Algorithms, Morgan Kaufmann, San Mateo, CA, 1989, pp. 10–19.
57 S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, Optimization by simulated annealing, *Science*, 220 (1983) 671–680.
58 P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht, 1987.
59 E.H.L. Aarts and J.H.M. Korst, *Simulated Annealing and Boltzmann Machines A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley, Chichester, 1989.
60 L. Davis (Editor), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.
61 J.H. Kalivas, Optimization using variations of simulated annealing, *Chemometrics and Intelligent Laboratory Systems*, 15 (1992) 1–12.
62 I. Rechenberg, *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart-Bad, 1973.
63 I. Rechenberg, The evolution strategy. A mathematical model of Darwinian evolution, in E. Frehland (Editor), *Synergetics — From Microscopic to Macroscopic Order*, Springer-Verlag, Berlin, 1984, p. 122.
64 H.-P. Schwefel, Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution, *Annals of Operations Research*, 1 (1984) 165.
65 T. Bäck, F. Hoffmeister and H.-P. Schwefel, A survey of evolution strategies, in R.K. Belew and L.B. Booker (Editors), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 2–9.
66 D.B. Fogel, An evolutionary approach to the traveling salesman problem, *Biological Cybernetics*, 60 (1988) 139.
67 D.B. Fogel and J.W. Atmar, Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems, *Biological Cybernetics*, 63 (1990) 111.
68 D.B. Fogel, L.J. Fogel and V.W. Porto, Evolving neural networks, *Biological Cybernetics*, 63 (1990) 487.
69 L.J. Fogel, A.J. Owens and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.
70 M. Herdy, Application of the evolutionsstrategie to discrete optimization problems, in H.-P. Schwefel and R. Manner (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer, Berlin, 1991, pp. 188–192.
71 J.H. Holland, K.J. Holyoak, R.E. Nisbett and P.R. Thagard, *Induction Processes of Inference, Learning and Discovery*, MIT Press, MA, 1986.
72 D.E. Goldberg and J.H. Holland, Guest editorial: Genetic algorithms and machine learning, *Machine Learning*, 3 (1988) 95–99.
73 L.B. Booker, D.E. Goldberg and J.H. Holland, Classifier systems and genetic algorithms, *Artificial Intelligence*, 40 (1989) 235.
74 R.E. Smith and D.E. Goldberg, Reinforcement learning with classifier systems, *IEEE Transactions on Systems, Man, and Cybernetics*, 16 (1990) 184
75 S. Forrest and J.H. Miller, Emergent behavior in classifier systems, *Physica D*, 42 (1990) 213.
76 M. Compiani, D. Montanari and R. Serra, Learning and bucket brigade dynamics in classifier systems, *Physica D*, 42 (1990) 202.
77 D.E. Goldberg, Computer-aided pipeline operation using genetic algorithms and rule learning. Part I: Genetic algorithms in pipeline optimization, *Engineering with Computers*, 3 (1987) 35.
78 D.E. Goldberg, Computer-aided pipeline operation using genetic algorithms and rule learning Part II: Rule learning control of a pipeline under normal and abnormal conditions, *Engineering with Computers*, 3 (1987) 47.
79 D.E. Goldberg and C.H. Kuo, Genetic algorithms in pipeline optimization, *Journal of Computers in Civil Engineering*, 1 (2) (1987) 128.
80 S. Forrest and A.S. Perelson, Genetic algorithms and the immune system, in H.-P. Schwefel and R. Manner (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1991, pp. 320–325.
81 B. Manderick, Selectionist categorization, in H.-P. Schwefel and R. Manner (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer, Berlin, 1991, pp. 326–330.
82 J.H. Holland, Personal communication, University of Michigan, 1989.
83 H. Atlan and I.R. Cohen (Editors), *Theories of Immune Networks*, Springer, New York, 1989.
84 H. Bersini and F.J. Varela, Hints for adaptive problem solving gleaned from immune networks, in H.-P. Schwefel and R. Manner (Editors), *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1991, pp. 343–354.
85 H. Bersini and F.J. Varela, The immune recruitment mechanism: A selective evolutionary strategy, in R.K Belew and L.B. Booker (Editors), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 520–526.
86 D.E. Rumelhart, J.A. Feldman and P.J Hayes (Editors), *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1986.
87 Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.
88 L. Steels, Artificial intelligence and complex dynamics, AI MEMO 88-2, AI Lab, Vrije Universiteit, Brussel, January 1988.
89 P.G. Wolynes, Chemical reaction dynamics in complex molecular systems, in L. Stein (Editor), *Lectures in the Sciences of Complexity*, Addison-Wesley, CA, 1989, p. 355.
90 S. Rasmussen, C. Knudsen, R. Feldberg and M. Hindsholm, The coreworld: Emergence and evolution of cooperative structures in a computational chemistry, *Physica D*, 42 (1990) 111.

91 J Byl, Self-reproduction in small cellular automata, *Physica D*, 34 (1989) 295
92 W B Dress, Synthetic organisms and self-designing systems, in *Proceedings of 1989 Goddard Conference on Space Applications of Artificial Intelligence, Goddard Space Flight Centre, Greenbelt, MD*, NASA, 1989, p 335
93 C G Langton (Editor), *Artificial Life Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, Addison-Wesley, 1989
94 C G Langton, Computation at the edge of chaos Phase transitions and emergent computation, *Physica D*, 42 (1990) 12
95 H Haken, *Information and Self Organisation*, Springer, Berlin, 1988
96 C B Lucasius and G Kateman, GATES Genetic Algorithm Toolbox for Evolutionary Search, Software library in ANSI C, Laboratory for Analytical Chemistry, Katholieke Universiteit Nijmegen, January 1991
97 D Whitley, The GENITOR algorithm and selection pressure Why rank-based allocation of reproductive trials is best, in J D Schaffer (Editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989, pp 116–121
98 L Davis (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991
99 S E Bayer and L Wang Problem solving with genetic algorithms and Splicer Technical Report AIAA 91-3836-CP, American Institute of Aeronautics and Astronautics, 1991

CHAPTER 2

Understanding and Using Genetic Algorithms. Part 2. Representation, Configuration and Hybridization

Contents

Understanding and Using Genetic Algorithms. Part 2. Representation, Configuration and Hybridization

Abstract

Starting from Part 1, Part 2 of this tutorial presents a digested compilation of pragmatic concepts and commonly applied techniques concerning genetic algorithms. The aim of this treatise is to support the novice practitioner in choosing a representation, a configuration, and, possibly, a hybridization technique for the genetic algorithm applied to the problem of interest – hence, to contribute to a unifying conceptual framework of general importance for the application of genetic algorithms.

1 Introduction

Any implementation of a genetic algorithm comprises two main ingredients: a population of strings that represent candidate solutions for the target problem, and heuristics that manipulate these strings in search of the true solution.

This tutorial part starts with a general introduction to problem representation. Thereby, examples related to chemical problems are given that should facilitate further reading.

Next, a generic flowchart for genetic algorithms is given. Its constituent modules are categorized according to a number of practically relevant properties and criteria. One of the presented taxonomies, for instance, underscores that some modules are engaged with exploiting currently available information in the search process, whereas other modules are engaged with exploring the search space; it is explained why the former are to a large extent problem-independent, while the others are confined to certain problems or problem types.

The remainder of this tutorial part first treats exploration heuristics for different problem types, followed by a discussion on commonly used exploitation heuristics. Following this, a number of approaches towards optimizing configuration are outlined. Finally, a taxonomy of hybridization strategies aimed at the enhancement of performance is provided.

The conclusion summarizes important general properties of genetic algorithm methodology.

2 Problem representation

In this section, we illustrate by way of some examples how candidate solutions for a particular problem might be represented as strings; recall from Part 1 [56] that the problem representation defines the search space.

For this discussion, it is irrelevant how candidate solutions are evaluated. It is, however, important to realize that different representations for the same problem result in search spaces which differ merely in the way the candidate solutions therein are distributed (i.e. how far they are mutually separated), and that there is no unique answer to the question which problem representation is best; the answer, as pointed out in Part 1 [56], depends on, inter alia, the way in which the strings are manipulated (reproduced and modified) by the evolutionary heuristics, and on the nature of the problem, in general.

In passing, it is increasingly argued in the mainstream literature on genetic algorithms that datastructures more complex than strings – e.g. trees structures, two- or higher dimensional arrays, etc. [62] – probably constitute more appropriate representations of candidate solutions for certain problems. Although such arguments seem plausible, we will nonetheless refer to any kind of encoded candidate solution as "string", hereafter – without loss of generality, in fact, because any datastructure (no matter how complex) is ultimately stored in a linear array of computer memory.

The next subsections outline representations for numerical parameter estimation problems, subset selection problems, and sequencing problems – abbreviated as NUM, SUB and SEQ problems, re-

spectively (see Part 1 [56]).

2.1 Numerical parameter estimation problems

The first example concerns the problem of finding an optimal three-dimensional macromolecular structure, or conformation, e.g. of proteins [72] or DNA [49, 50, 45, 6]. It is known that by far most of the flexibility in a molecule resides in torsional rotations along single bonds. Therefore, a sensible representation of a candidate conformation would be a string of real numbers, each of which is the value proposed for a particular torsion angle. For instance,

```
135.0  045.0  315.0  000.0  090.0
```

represents a candidate conformation involving 5 torsion angles with values expressed as degrees. (The additional space in the illustration is only used for clarity, i.e. it is not actually part of the string.) However, many other representations are conceivable. For instance, a decimal (integer) representation may be convenient: decimals from 0 up to some maximum, ρ, are used such that these correspond with equidistantly chosen levels on the real range $[0.0°, 360.0°)$ according to a digitization relation: $x = \frac{D}{\rho} \cdot 360.0$, where x is a digital real level, D the corresponding decimal-encoded value and ρ is the encoding resolution; due to the digitization, the search space is basically a search grid in the real space spanned by the torsion angles. For $\rho = 8$, the example candidate conformation can then be represented as the string:

```
3  1  7  0  2
```

The integers may be expressed in a non-decimal numerical base as well. For instance, in numerical base 2, one obtains binary strings, or bitstrings, as representations of candidate conformations. Two types of *binary encoding* are commonly used: *regular* binary encoding and *Gray* binary encoding. In regular binary encoding, the example candidate conformation is represented as the bitstring:

```
0 1 1  0 0 1  1 1 1  0 0 0  0 1 0
```

In Gray binary encoding, the example candidate conformation is represented as the bitstring:

```
0 1 0  0 0 1  1 0 0  0 0 0  0 1 1
```

Gray binary encoding and its generally assumed advantages over regular binary encoding, are discussed in Section 5.2.1 below.

2.2 Subset selection problems

The second example concerns the problem of selecting a subset of items from a given set of items, the so-called source set. The items may be: functional groups in a molecule that play a role in an IR spectrum interpretation problem; or features that play a role in a pattern recognition problem [73]; or sensors (detectors) in a multi-sensor array that play a role in a wavelength selection problem for multi-component analysis [52, 46, 48]; or centra that play a role in a k-medoid clustering problem [51]; etc. In general, subset selection problems can be categorized according to whether a fixed-size subset or an unknown-size subset must be found.

Items are indicated by unique identity labels, or elements, e.g. alphabetical letters or integral numbers; in the latter case, the elements are called indices. Using elements, two types of *subset encoding* are considered here: *binary* subset encoding and *direct* subset encoding.

In the former, a candidate subset is represented as a bitstring that serves as a template to mask the source set's elements put in some chosen (i.e. fixed) sequence; in the template, then, a "1" denotes "select the element at that position" and a "0" denotes the opposite. For instance, in

```
3 1 5 2 9 7 8 4 0 6
0 1 1 0 0 1 0 0 1 0
```

the bottom string represents a candidate subset that proposes the selection of the subset {1,5,7,0} from the source set {0,1,2,3,4,5,6,7,8,9}. Note that if the problem concerns fixed-size subset selection, then the number of "1"s in the bitstrings must remain constant upon modification (i.e. upon recombination or mutation). How this encoding constraint is heuristically accomplished depends on the creativity of the designer of the modification operator concerned. For instance, as a recombination operator one may naively apply simple recombination – **1X** (Part 1 [56]) – followed by a patching algorithm that compensates for any incidentally arising shortage/excess of 1-bits in a resulting child string by flipping an appropriate amount of randomly selected 0/1-bits therein; in this way, the patching algorithm acts as a specialized, built-in mutation operator, which may obviate any other mutation operator.

In direct subset encoding, on the other hand, a candidate subset is represented directly in terms of

he elements; then, the example candidate subset an be represented as the string:

1 5 7 0

Three important remarks can be made about direct subset encoding. First, it is constrained in hat each element is not allowed to occur more han once; modification operators can be designed uch that this encoding constraint is satisfied; an example is given in Section 5.4.2 below. Second, if the problem is about unknown-size subset election, then the genetic algorithm used must additionally include provisions for variable-size trings in the population. Third, the representation seems redundant, because 23 other strings besides 1 5 7 0 – e.g. 5 0 7 1 – represent the same subset; it should be realized, however, that his reflects human perception, i.e. not necessarily the interpretation of the modification operators n the genetic algorithm used; besides, duplicate candidate solutions in the population are not forbidden (and in fact not uncommon, in general, especially when convergence is approached).

There also exist subset selection problems wherein the task is not only to select the optinal subset of items, but also to determine their optimal order. This special kind of subset selection problem is referred to as a sequenced subset selection problem (see Section 5.4.2 below). Here, direct subset encoding is more appealing han binary subset encoding: the latter encoding seems simply unsuitable to encode order information: moreover, the former encoding does not seem redundant.

2.3 Sequencing problems

The third example concerns the problem of finding an optimal order of items given. In scheduling problems, the items are pre-defined actions that can be arranged in different orders to obtain different schedules; an example in chemical engineering s outlined in [10]. In routing problems, the items are fixed objects that can be connected in different orders to obtain different routes, e.g. the traveling sales-person problem. A routing problem of chemcal interest is described in [59]: the problem is to ind an optimal order of chromatographic columns, using selectivity for the components as an optinality criterion; the order affects selectivity due to tailing effects and other non-ideal behavior of the chromatographic process.

A sensible representation for a candidate sequence is simply a permutation of (alphabetical or numerical) elements, e.g.

5 0 4 9 2 6 7 1 8 3

Note that in this representation, or *permutation encoding*, each element must occur exactly once; modification operators can be designed such that this encoding constraint is satisfied; some examples are given in Section 5.5.1 below.

3 Generic flowchart

In Part 1 [56], the flowchart of a so-called simple genetic algorithm was discussed. In general, a genetic algorithm may be considered any algorithm that fulfils a general, more or less informal description of the evolutionary problem solving concept that resides in the mainstream literature on the topic. By this loose definition, innumerable flowcharts legally represent genetic algorithms, in principle. Our version of abovementioned general description is hereafter called the *generic flowchart*:

DISCUSSION OF THE GENERIC FLOWCHART
(negative-indexed steps represent preliminary actions):

−3. Parameterize the objective function (module 0):
The domain parameters incorporated in the objective function are set; these values are derived from domain-dependent input data, e.g. an experimental spectrum.

−2. Parameterize the genetic routines (modules −1, 1, 2, 3, a, b, 4):
Control parameters are set, using domain-independent input data. Examples of such parameters are the recombination rate, the mutation rate and the population size; if domain-dependent genetic routines are used as well, then their domain parameters, too, are set, using domain-dependent input data.

−1. Initiate strings:
The initial population is created. Each string

in the population uniquely represents (encodes) a candidate solution of the target problem; many encodings are possible for a particular problem, and the one chosen reflects an educated guess of the implementor. Without prior knowledge available, the initial population usually comprises strings that represent random candidate solutions. The population size is specified by the user (in module **-2**). Often, the strings are of a fixed length; this is for instance the case in a simple genetic algorithm.

0. Evaluate strings:
Each string in the population is decoded to obtain its actual meaning. This is then passed on to the objective function, ϕ, in order to calculate and store a numerical value indicating performance, called the raw fitness, $f_0 = \phi$. It is hereby assumed, without loss of generality, that ϕ is a maximization criterion; if this is not the case, then f_0 is calculated by applying some inverting transformation to ϕ, e.g. sign inversion ($f_0 = -\phi$) or reciprocation ($f_0 = 1/\phi$). In this way, any genetic algorithm always explicitly maximizes f_0. (This measure is called "raw" to emphasize how close it stands to the problem domain, having undergone minimal numerical transformation.)
Next, a termination criterion is applied, based on either the observed performance or on a maximum generation. If it succeeds, the algorithm terminates; otherwise it continues.

1. Scale fitnesses:
Raw fitnesses are scaled in a number of (possibly 0) transformation steps when they are not suitable as such for the selective reproduction or -replacement procedure (modules **2** and **4** below). Linear scaling is employed in a simple genetic algorithm and many other implementations.
Fitness scaling is not always needed. In particular, this is the case when the reproduction- and replacement procedure do not process fitness values, but only fitness ranks; both processing modes are examined closer below.

2. Selectively reproduce strings:
A selection criterion that shows bias for strings with comparatively high (scaled) fitness values is iteratively applied to the strings in the population; each selected string is copied to a temporary (new) population, subject to a size not larger than the current population. Due to the ensuing selection "pressure", the temporary population may be expected to perform better than the current population. In a simple genetic algorithm and many other implementations, the strings are reproduced probabilistically with expected rates proportional to their (scaled) fitness; this strategy has become known as reproduction by roulette selection; in Part 1 [56] another, equivalent, metaphor was used: biased die selection.

3. Pair strings:
In preparation for recombination (module **a** below), the strings in the temporary population are paired to obtain parent pairs. The pairing is either random (as in a simple genetic algorithm) or according to a more sophisticated criterion.

a. Recombine strings:
The recombination operator is iteratively applied to the pairs of parent strings in the temporary population, to each with probability p_r (the recombination probability). Upon success for a particular pair, equally sized string fractions ("partial candidate solutions") are selected and swapped so that new strings are formed in the temporary population; the fractions are selected probabilistically, ideally subject to the so-called building block principle [29, 56]: sites for string fractioning are selected such that disruption (loss) of important information is minimal, i.e. preservation of important information is maximal; by assembling new strings from "good old parts" it is intended that comparatively large steps in performance, towards the true solution, are made.
Upon failure, the child strings simply become copies of the parent strings in most cases.

b. Mutate strings:
The mutation operator is iteratively applied to the strings in the temporary population. In a particular string, random local changes are imposed, each with probability p_m (the muta-

tion probability). Mutation, too, introduces diversity into the temporary population, but merely with the aim to enhance the productivity of recombination.

4. Selectively replace strings:
The strings in the temporary population replace an equal number of strings in the current population; the latter are selected according to some criterion, e.g. one with bias for the worst performing strings. In a simple genetic algorithm, the size of the temporary population equals that of the current population, leaving complete replacement – or replacement without "generation gap", or generational population handling – as the only possibility.

Return to **0**.

In the further discussion, we will not be much concerned with the preliminary actions (steps **−3**, **−2**, **−1**); traditionally, these are paid minor attention in other literature too. Thus, our attention is focused mainly on the evolution cycle, graphically summarized in Figure 1.

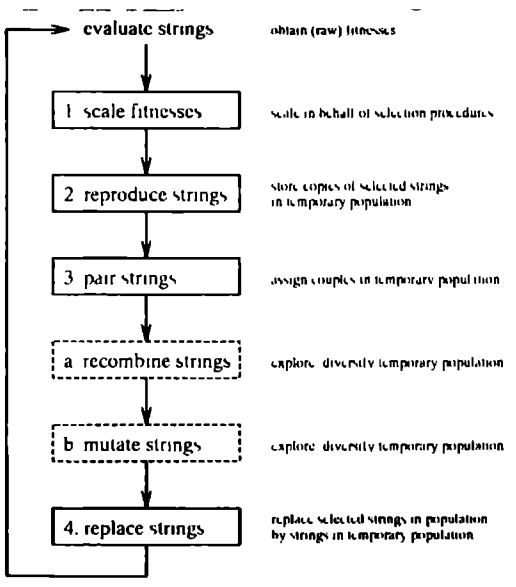

Figure 1: Generic evolution cycle in genetic algorithms: solid boxes and dashed boxes represent modules engaged with exploitation and exploration, respectively.

Specific routines for the modules in the evolution cycle are targeted to the population and/or temporary population. They iterate special subroutines – called operators – targeted to individual strings. For instance, a fitness scaling routine (module **1**) iterates an operator, called a fitness scaling function, across the vector of fitnesses for the strings in the population. A reproduction routine and a replacement routine (modules **2** and **4**) iterate an operator, called a selection criterion, across the strings in the population. A pairing routine (module **3**) iterates an operator, called a pairing criterion, across the strings in the temporary population. A recombination routine (module **a**) iterates a recombination operator across the string pairs assigned in the temporary population. A mutation routine (module **b**) iterates a mutation operator across the strings in the temporary population.

During a run of a genetic algorithm, one normally monitors the best candidate solution in the successive populations and keeps the best-up-to-present saved. In this way, after termination, the best candidate solution ever encountered during the run is known. The practitioner may also be interested in knowing other candidate solutions that came up during the search, if that would provide additional information according to his/her insights.

4 General notes on configuration

The instantiation of the modules comprising the generic flowchart – i.e. the effort which amounts to the selection, implementation and parameterization of the constituent procedures, or routines – is referred to as genetic configuration. (Hereafter, the adverb "genetic" is regularly used as a shorthand to indicate "concerning genetic algorithms".) That is to say, genetic configuration results in pinpointing a specific genetic algorithm, dedicated to a particular target problem, within the family of genetic algorithms spanned by the generic flowchart. That this family is vast, should be evident. For, one can already conceive of innumerable routines as specific implementations for each module; the number of combinations must be even larger. Summarizing, owing to its tremendous configurational flexibility, or *versatility*, ge-

netic methodology has the ability to adapt to very divergent problems.

In building a particular genetic application, the practitioner will at first be inclined to recourse to an arsenal of problem representations, recombination routines, mutation routines, selection routines, fitness scaling routines, etc., that have, in the course of time, been designed and documented in the literature by experts. Only when the application requires unprecedented heuristics, the practitioner will need to design the necessary genetic routines from scratch – thereby, indeed, extending the arsenal. The design of new recombination heuristics normally entails the major effort, because the building block principle (and, possibly, encoding constraints) have to be complied with. He/she can accomplish this by starting from intuitively plausible assumptions about partial solutions – how they look like in terms of a particular representation, and how they should be combined to create better (larger) partial solutions; often, a few alternative approaches are considered as well. Ultimately, these educated guesses should result in a recombination heuristic that is capable of processing an appropriate type of building block for the search task concerned.

The next subsections categorize genetic routines and their effects according to a number of properties and criteria of practical importance, and explain why finding an optimal configuration is difficult.

4.1 Elementary heuristic actions

The constituent routines of a genetic algorithm that handle the strings in the population, must make at least one assumption about the problem representation: the string length. Some of these routines make more representational assumptions, such depending on the particular heuristic actions they perform. Other genetic routines, on the other hand, are not (directly) preoccupied with strings at all, e.g. most fitness scaling routines.

Four elementary types of heuristic actions on strings can be distinguished. These are listed in Table 1, along with the genetic module(s) that perform them. String copying and -deletion are straightforward actions and thus need no further explanation. Details as to how string modification and -interpretation are performed, follow from the description of the pertinent routines in sections below.

ACTION	MODULE
copy string	**2**
delete string	**4**
modify string	**a**, **b**
interpret string	**a**, **b**, **0**, (**1**, **2**, **3**, **4**)

Table 1: Overview of elementary heuristic actions, and the modules in the flowchart performing these: for modules listed between parentheses, it depends on the specific implementation whether the corresponding action is undertaken.

4.2 Exploitation and exploration

The modules in the generic flowchart can also be categorized according to whether they explore the search space or whether they exploit supposedly useful information that has accumulated in the population during the search yet to be completed.

The modules **1**, **2**, **3** and **4** are called exploitation modules, as they affect the survival rates of the strings in the population and thus are concerned with the utilization of important information currently stored in the population, as indicated by fitness. The specific routines for these modules do not modify the strings and the majority of them perform at most weak interpretation of the strings. They can therefore be chosen (almost) independently of the type of target problem.

The modules **a** and **b**, on the other hand, are called exploration modules, as they modify the strings in the population and thus perform steps in the search space. Specific routines for these modules perform at least moderate interpretation of the strings. They are therefore confined to a specific class of problems or possibly even to a single problem.

4.3 Deterministic- versus probabilistic routines

Genetic routines are either deterministic or probabilistic. For an explanation of these properties, it is convenient to define the concepts "reach" and "total reach".

The reach of a routine is understood to be the collection of possible outcomes obtained for a par-

ticular legal input; e.g. the reach of a mutation operator is the collection of strings it can produce, given a particular string as input. The total reach of a routine is understood to be the collection of possible outcomes obtained for all legal inputs; e.g. the total reach of mutation operators is the search space, as a rule.

A routine is called deterministic if, for an arbitrary legal input, repeated calls produce the same outcome, i.e. if the reach is one outcome. A routine is called probabilistic (or stochastic) if, for an arbitrary legal input, repeated calls may result in different outcomes according to a probability distribution across the total reach; in this case, the reach consists of all outcomes in the total reach that feature non-zero probability. In the special case that the probability distribution is uniform, one speaks of a random routine.

Among commonly applied exploitation routines, one encounters deteministic- as well as probabilistic members. Commonly applied exploration operators, on the other hand, are probabilistic, as a rule.

4.4 Principal conditions for optimal performance

Recapitulating from Part 1 [56], the principal conditions for optimal performance can be summarized as:

1. The exploration is intelligent;
2. The exploration and exploitation are balanced.

4.4.1 Intelligent exploration

An exploration operator is called intelligent when it can modify strings to a great extent at little loss of relevant implicit information – or properties, or schemata – stored in the strings. Recall from Part 1 [56] that such intelligent exploration is traditionally achieved by the recombination operator. (We say "traditionally", since, according to a recent study [74], mutation can always be designed such that it acts in a way effectively similar to recombination; we discard this novel development here.)

As the recombination operator takes as input a pair of parent strings, cuts them into fractions, and exchanges some of these fractions between the strings to produce two new, equally sized strings (child strings), it preserves, as opposed to the mutation operator, all constituent elements of the original string pair. Intuitively, this is indeed an important minimal condition to preserve good (i.e. highly fit) schemata, or building blocks, from both parent strings. In this way, recombination exhibits assembly behavior and thus can accomplish – better so than mutation – comparatively large leaps in progress, even when the operation does not show bias for any particular property. Importantly, with bias for the good properties, the recombination operator is even more intelligent; recall from Part 1 [56] that this notion is commonly referred to as the *building block principle*. Of course, this principle requires that the designer of the recombination operator knows what the good properties for the given type of search task are. If such knowledge is based on false assumptions, the search can be frustrated (i.e. deceived) and performance can thus be seriously degraded, as was also pointed out in Part 1 [56].

It is important to realize that, in general, any bias in a particular exploration operator essentially comes down to some form of exploitation – be it useful or not.

4.4.2 Balancing exploration and exploitation

Exploration and exploitation are the co-operating antipoles in evolutionary search. The required balance in the trade-off between exploration and exploitation is achieved when efficient exploitation takes place while sufficient diversity is maintained in the population for productive exploration. If the proportion is too much in favor of exploration, population diversity decays too slowly and the search tends to become blind. If the proportion is too much in favor of exploitation, population diversity decays too rapidly, provoking premature convergence, hence blocking schema processing – another notorious failure mode.

The effect of a single configurational factor on the balance between exploration and exploitation is not difficult to understand, in general; for instance, an increase in the recombination probability (p_r) or mutation probability (p_m) will increase the exploration/exploitation ratio. However, the effect of several configurational factors simultaneously is generally not easy to predict, which may

be ascribed to the fact that many of these factors are strongly correlated (see Section 4.5 below).

The virtue of rank-based selection. In maintaining a balance between exploration and exploitation during a run, the choice of the reproduction- and replacement procedure is crucial. In particular, it is important whether such a procedure selects strings on the basis of fitness value or -rank.

In a simple genetic algorithm and many other traditional implementations, strings are reproduced probabilistically with expected rates proportional to their raw fitness value. At first glance, this seems reasonable: if individual A is 6 times fitter than individual B, then why should A not reproduce at a 6 times higher rate than B? (As for replacement, a similar argument may be put forward, provided that the role of fitness is reversed in some manner.) This reasoning is, however, objectionable by the following arguments.

The relative reproduction rates depend on the choice of the objective function for the target problem. Indeed, whereas the objective function uniquely defines the target problem, the converse is certainly not necessarily true. For instance, in a curve fitting problem, several plausible measures for the distance between a candidate curve and the target curve spring to mind. The fact that one objective function for a particular problem may lead to better performance than another for the same problem – assuming fitness-proportional reproduction, or fitness value-based reproduction, in general - has lead to an abundance of fitness scaling functions (see Section 6.1 below); for, a fitness scaling function may be regarded as essentially an extension to an objective function such that another objective function is obtained for the same problem.

Although different objective functions for the same problem are generally in disagreement about (ratios of) fitness values, this is usually not the case when fitness ranks are concerned: if one objective function ranks individual A fitter than individual B, then, normally, another objective function for the same problem produces the same judgment; for one thing, consistency demands that all objective functions for the problem assign the highest rank to the true solution. Hence, the theoretical advantage of rank-based selection seems to be that it lacks the ambiguous nature of value-based selection. — Grefenstette and Baker [38] add to these arguments the premise that rank-based selection seems closer to the mechanism of natural selection in that if one individual is fitter than the other, it often does not matter how much fitter exactly it is. Further, Whitley [87] argues (and corroborates by experiments) that a ranking strategy is in some ways even most consistent with the schema theorem, because it is non-parameteric; that is, a ranking strategy makes it unnecessary to introduce additional parameters that are not accounted for by the schema theorem in order to control selective pressure.

In numerous empirical studies, practitioners have observed that, in maintaining a balance between exploration and exploitation during a run, rank-based selection is often more robust than fitness-proportional selection. In the light of the foregoing, this stability can be explained as follows: when there are strings in the population that have far above-average fitness (such as is likely to happen in the early stages of the search), rank-based selection resists premature convergence. On the other hand, when differences in fitness are small (such as is normally the case in the late stages of the search), rank-based selection encourages competition.

Rank-based selection should not be considered a panacea, though, since there also exists a considerable body of empirical evidence in favor of selection on the basis of (scaled) fitness values rather than -ranks.

4.5 The configuration bottleneck

Structurally simple as genetic algorithms may seem and intuitively appealing their underlying evolutionary principles, the overall evolution process is mechanically very complex due to strong correlations between configurational factors. These are caused by the intensive procedural interactions that take place through the population – the common target of all procedural actions. Combining this fact with the aforementioned versatility of genetic algorithms, it is not difficult to understand why the search for an adequate (let alone optimal) configuration, can be a troublesome enterprise for which, not surprisingly, no standard methodology is available. (The task becomes even more difficult when a more complex target problem is concerned.) For this rea-

son, we say that applied genetic methodology is poorly accessible. Indeed, as was pointed out in Part 1 [56], the best configuration strategies actually appear to be those based on intuitive insight obtained from empirical experience; some practical rules of thumb – mostly involving the control parameterization side of configuration (e.g. [37]) – are available in the mainstream literature and provide some guidance. Other, criterion-based, configuration strategies used in practice are discussed in Section 7.1 below.

An efficient and efficacious way for the novice practitioner to overcome the poor accessibility of applied genetic methodology, resides in training practice with genetic software [54] – starting with available education software, then progressing towards more advanced software.

4.6 Application scope

The versatility of genetic algorithms, or, more precisely, the vast collection of problems amenable to a genetic approach, may be called the overall application scope of the generic flowchart.

When genetic routines are specified so that the generic flowchart is (partially) instantiated, the application scope is reduced, i.e. lies within the application scope of each of the constituent routines. An important kind of partial instantiation results in what has been called a *genetic shell* [54, 55]: a flowchart wherein all routines, except the objective function (and the input routine that parameterizes it, module **–3** in the flowchart), are specified.

We have found it useful to categorize the application scope of genetic routines according to three hierarchical size classes: narrow, medium and wide; these correspond with strong, moderate and weak implicit assumptions about the target problem, respectively (Table 2); the strength of assumptions is indicated by the extent of string interpretation. (A similar classification for the gamut of problem solving heuristics from "weak" to "strong", was briefly outlined in Part 1 [56]; for a more detailed discussion on this issue, the reader is referred to the excellent treatise due to Newell [63].)

Objective functions (module **0**) have an application scope as narrow as one problem – the target problem, by definition. For this reason, objective functions are called "domain-dependent".

Exploitation routines (modules **1**, **2**, **3** and **4** in the flowchart) have the widest application scope. This may be attributed to the fact that they do not modify the strings, but merely manipulate their survival rates. With no direct relation to the target problem, exploitation procedures are called "domain-independent"; their choice is chiefly motivated by the practitioner's considerations concerning the (stability in the) convergence rate.

Exploration routines (modules **a** and **b**), as they modify strings, can not avoid making at least moderate assumptions about the target problem; this is especially true for recombination operators, since these incorporate the building block principle. Accordingly, exploration routines can not possibly be suitable for all problems. For instance, simple recombination, **1X**, in a genetic algorithm for sequencing problems – using permutation encoding – can not guarantee that encoding constraints will not be violated, as can be verified easily by the following example:

$$
\begin{array}{lcccccccccccc}
P_1 & = & 8 & 9 & 7 & 0 & 5 & 1 & \vert & 4 & 2 & 6 & 3 \\
P_2 & = & 4 & 9 & 1 & 8 & 6 & 7 & \vert & 5 & 0 & 3 & 2
\end{array}
$$

$$\downarrow \mathbf{1X}$$

$$
\begin{array}{lcccccccccccc}
C_1 & = & 8 & 9 & 7 & 0 & 5 & 1 & 5 & 0 & 3 & 2 \\
C_2 & = & 4 & 9 & 1 & 8 & 6 & 7 & 4 & 2 & 6 & 3
\end{array}
$$

where P_1 and P_2 are legal parents strings (since they are permutations), "|" indicates the breakpoint, and C_1 and C_2 are illegal child strings (since they are not permutations). Indeed, **1X** is aimed at other problems in practice, predominantly numerical parameter estimation problems, which are free of encoding constraints of the kind above.

Traditionally, exploration routines are used that make only moderate assumptions about the target problem. Such exploration routines are called "domain-independent", even though their application scope is not wide, but medium. Genetic algorithm applications that incorporate domain-independent exploration routines are usually developed from genetic shells, because that minimizes the development time and -cost.

Since recently, domain-dependent exploration routines – based on strong assumptions about the target problem – meet wider adherence [14, 61]; an example of this is the incorporation of established local search heuristics (see Section 8.6 below). In comparison with domain-independent exploration routines, the use of domain-dependent

PART OF FLOWCHART	DOMAIN ASSUMPTIONS	APPLICATION SCOPE
objective function	strong	narrow
exploration routines	moderate strong	medium – narrow
exploitation routines	weak	wide

Table 2: A taxonomy for the routines in the evolution cycle in a genetic algorithm (Figure 1), based on application scope.

exploration routines usually leads to better performance (faster convergence towards acceptable solutions); such a trade-off between the application scope and search efficiency of search heuristics is quite common in optimization practice, as was pointed out in Part 1 [56]. However, genetic algorithm applications that incorporate domain-dependent exploration routines are comparatively expensive in terms of development time and -cost.

4.7 Problem taxonomy

In this section, we present two problem taxonomies: one according to problem type and another according to problem specificity. Both taxonomies are natural offshoots of the above concept of application scope for genetic routines.

4.7.1 Problem type

Any "domain-independent" exploration routine generally addresses a group of problems that have certain characteristics in common. Recapitulating and extending from Section 2 above, we distinguish three problem categories, or -types, of practical importance to analytical-chemical problem solving (and beyond):

NUM – numerical parameter estimation;

SUB – (either fixed-size or unknown-size) subset selection;
this type of problem is a special case among partitioning, or grouping, problems: a set needs to be partitioned into effectively two subsets (binary partitioning problem).

SEQ – sequencing;
this type of problem is a special case among graph configuration problems: a one-branch graph needs to be found.

SUB and SEQ problems belong to the more general class of combinatorial problems. Incidentally, it should be borne in mind that the above problem taxonomy, albeit wide in scope and practically sensible, is not exhaustive nor unique.

Since their inception, genetic algorithms have been applied to solve NUM problems. Applications for SEQ problems have joined the scene since the mid-1980ies, to be followed by the first few applications for SUB problems a few years later; as analytical chemistry is concerned considerably with SUB problems, traditionally, it may contribute significantly to future applications of this kind.

Exploration operators for NUM, SUB and SEQ are discussed below. (Discussions on exploration operators for graph configuration problems – tree finding, in particular – can be found in e.g. [43]. Discussions on exploration operators for partitioning problems can be found in e.g. [5, 44, 40, 24, 23].) In the remainder, most attention will be paid to exploration operators for NUM problems, as these have the longest tradition of investigation and are attractive for beginners in that they can be approached successfully with comparatively simple representations and exploration heuristics.

4.7.2 Problem specificity

Within problem types, it makes sense to define subcategories. One may distinguish "problems" as instances of a problem type; for example, a wavelength selection problem is an instance of the SUB problem type. Thus, different problems of a particular problem type may use the same domain-independent exploration routines, but they demand different objective functions, in general (as the objective function depends on the target problem).

Furthermore, one may distinguish "problem instances" as versions of a problem. A problem instance is uniquely determined by the input data used by the problem's objective function; for ex-

ample, an instance of a wavelength selection problem is obtained for a particular UV spectrum used as the input data. Thus, different instances of a particular problem may use the same objective function, but the latter then uses different input data, in general.

4.8 Interplay between exploration and exploitation revisited

The interaction between exploration and exploitation is determined by the way in which these processes both play an important role in *schema processing* and the *population diversity decay*. Understanding these roles, therefore, constitutes a requisite for intuitive insights as typically used by the skilled practitioner in dealing with the trade-off between exploration and exploitation.

4.8.1 Role of exploitation

The role of exploitation can be summarized as follows:

Exploitation, or selection pressure:

1. preserves building blocks (good schemata);
2. disposes of bad schemata;
3. reduces population diversity.

Note: these effects apply to each population update, and therefore become apparent only after one or more generations have elapsed.

Selection pressure is required to promote the survival of building blocks. Traditionally, the reproduction procedure plays a key role therein: good schemata are preserved as their instances – good strings – are encouraged to reproduce; the opposite applies to bad schemata and strings. It is, however, important to realize that each schema processing step here – a string reproduction – involves all, i.e. good and bad, schemata contained in the string concerned. — Recent studies [80, 82, 22, 20] indicate that it can also be meaningful to effectuate selection pressure through the replacement procedure; in that case, the population is only partially replaced in each generation, in such a way that bad schemata are disposed of, as their instances – bad strings – are encouraged to be replaced; the opposite applies to good schemata and strings. Of course, each schema processing step here – a string replacement – too, involves all schemata contained in the string concerned.

Population diversity must decay in order to promote convergence towards the global optimum. Different exploitation procedures and parameterizations thereof, generally result in different decay rates and stabilities therein.

4.8.2 Role of exploration

The role of exploration can be summarized as follows:

Mutation:

1. creates population diversity;
2. preserves, creates, and disposes of schemata – traditionally without any premeditated preference (bias) whatsoever.

Note: in fact, these effects apply to individual string updates, hence indeed to population updates as well; they become apparent within a generation.

Mutation creates population diversity, hence tempers the decay of the overall population diversity, which is necessary to oppose premature convergence. The amount of diversity created depends on the mutation probability, p_m, and on the heuristics of the mutation operator. Recall from Part 1 [56] that mutation is normally assigned a minor role relative to recombination; this is accomplished by using a low value for p_m, typically within the range $[0.001, 0.05]$.

Recombination:

1. requires population diversity;
2. creates population diversity:
3. preserves and creates building blocks (good schemata):
   the building block principle;
4. disposes of bad schemata.

Note: in fact, these effects apply to individual string updates, hence indeed to population updates as well; they become apparent within a generation.

In order to be productive, the recombination procedure requires population diversity. For, when both parent strings are identical, the child strings produced by recombination are copies of

their parents (hence, of each other), by definition; the search space is then not explored.

Productive recombination creates population diversity, hence tempers the decay of the overall population diversity, which is necessary to oppose premature convergence. The amount of diversity created depends on the recombination probability, p_r, and on the heuristics of the recombination procedure.

Preservation and creation of building blocks, and disposal of bad schemata, is accomplished through the implementation of the building block principle, as pointed out above. In passing, a methodology for automated finding of building blocks has recently become available – embodied in so-called messy genetic algorithms [32, 31]. Messy genetic algorithms perform run-time search for building blocks according to a predefined criterion. Although the idea behind messy genetic algorithms is attractive, they have up to now seen only sporadic application. Therefore, they are not considered in the further discussion.

5 Exploration

After a brief, general introduction, this section presents a number of commonly used exploration operators and related encoding issues, for NUM, SEQ, and SUB problems, respectively. Most attention is paid to recombination operators, as these traditionally play a dominant role in exploration.

5.1 Introduction

This section introduces some terminology and conventions that should facilitate the further discussion.

5.1.1 Principal properties of recombination

Different exploration operators, when applied to a particular set of arbitrary legal inputs, turn out to have a larger reach (Section 4.3) than others, on average. This fact is captured by the notion "exploratory power" [76]: the extent to which an exploration operator can reach other solutions in the search space, i.e. the ability to create population diversity. Incidentally, in the particular case of a recombination operator, the reach for a particular pair of parent strings is sometimes called the operator's "child set".

Besides exploratory power, other important properties of recombination operators are "disruptivity" (Part 1 [56]: the opposite of the tendency to preserve building blocks) and "recombinative potential" [76] (the ability to create new building blocks from other building blocks).

Ideally, a recombination operator should feature a high exploratory power, a low disruptivity, and a high recombinative potential. However, since these properties tend to be positively correlated, they can not be optimized simultaneously without making compromises. (The correlation is plausible, considering that schemata comprise of strings and *vice versa*.) Thus, for instance, a higher exploratory power tends to be accompanied by a higher disruptivity. (A note of caution is in place here: in the literature, statements on disruptivity are often based implicitly on the assumption that Holland schemata are processed; this is not always justified.) In earlier times, genetic algorithm scientists were not very much concerned with this dilemma, as they considered the building block principle the most important criterion for good performance, i.e. they emphasized minimizing disruptivity.

When disruption becomes a virtue. The exploratory power and recombinative potential of recombination operators have only recently been acknowledged as figures of merit in evolutionary search [80, 75, 76, 18, 22, 20], especially (1) in the late stages of the search process when the population is quite homogeneous, and (2) when the population size is too small to provide the necessary diversity for complex search spaces. Nonetheless, from a theoretical standpoint it is unacceptable to assume that search can be efficient if recombination constantly disrupts building blocks. The paradox is solved if one understands that a highly disruptive recombination procedure does not necessarily cause high overall disruption. For, building block preservation can be accomplished by selection pressure besides recombination, as pointed out above (Section 4.8.1). Hence, the disruptive impact of a highly explorative recombination procedure on building blocks can, in principle, be counterbalanced by promoting building block preservation through selection pressure. This is

important because it enables one to approach an optimal configuration, featuring a low overall disruption rate in spite of a high exploratory power and recombinative potential. On the other hand, however, it is important to realize that selection pressure – contrary to recombination – is not directly engaged in building block creation.

In view of the promising results that have recently been reported for highly disruptive recombination operators (e.g. **B.UX**, discussed in Section 5.2.2 below) under comparatively high selection pressure, it may be expected that practitioners will increasingly follow this approach.

5.1.2 Name conventions

In order to serve the purpose of brevity, we choose mnemonics as the names for recombination- and mutation operators. Thereby, the first letter in a name denotes the type of encoding (see Section 2 above) that underlies the operator concerned: **B** for binary encoding (applies to NUM operators), **D** for direct subset encoding (applies to SUB operators), and **P** for permutation encoding (applies SEQ operators). An example is **B MX** (binary multipoint recombination), discussed in Section 5.2.2 below.

The name of mutation operators ends in **M**. The name of recombination operators ends in **X**, graphically suggesting the crossover of two chromosomes – biological recombination.

5.1.3 Description

The description of a particular exploration operator is an account of the strategy according to which the characters, or digits, that compose the target string(s), are modified. In general, the modifications are aimed at so-called elements: groups of digits placed contiguously according to a predefined partitioning for all strings in the population. Possibly, these elements are simply single digits in the strings. By modifying larger elements – groups of digits – at a time, the exploration operator concerned can accomplish larger steps in the search space, which may be rewarding in some cases.

In describing exploration operators, so-called binary templates often turn out to be an intuitively pleasing aid, especially when complex recombination operators are concerned. A binary template is a bitstring in which the bits serve to mask the elements in the target string; elements masked by a 1-bit are considered as selected for modification. Incidentally, it is important to realize that a particular intuitively pleasing description of an exploration operator in terms of binary templates, does not necessarily correspond with the most efficient, equivalent implementation in software.

It turns out that virtually all existing recombination operators can be described in terms of merely two basic types of binary templates, namely: *multipoint binary templates* and *uniform binary templates*, discussed next.

Multipoint binary templates. A random multipoint binary template – denoted as $T_M(n)$, where n is an integer – is understood to be a bitstring which fulfils the following description: in scanning the template's consecutive bits from left-to-right (or *vice versa*), n inversions in bit value occur at random bit positions. A recombination operator that is described in terms of a $T_M(n)$ binary template, cuts each of two given parent strings at n randomly positioned breakpoints that correspond with the n bit value inversions in the template, into $n+1$ substring fractions that are used next in assembling the child strings. Two examples of a random 10-bit $T_M(2)$ binary template are:

0 0 0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 1 1 1

Within the family of random $T_M(n)$ binary templates, $T_M(1)$ (one-point) and, even more so, $T_M(2)$ (two-point) have met most frequent usage.

Uniform binary templates. A random uniform binary template – denoted as $T_U(n)$, where n is either an integer or fractional – is understood to be a bitstring which fulfils the following description: when n is an integer, the template contains exactly n 1-bits at random bit positions; otherwise, i.e. if n is fractional, then at each bit position the probability to encounter a 1-bit is n. Two examples of a random 10-bit $T_U(3)$ binary template are:

0 0 1 0 0 1 1 0 0 0
0 1 1 1 0 0 0 0 0 0

Two examples of a random 10-bit $T_U(0.3)$ binary

template are:

1 0 0 0 0 0 0 1 1 0
0 1 0 0 0 1 1 0 1 0

In many recombination operators that can be described in terms of a random $T_U(n)$, maximal exploratory power is achieved for $n = 0.5$ (see Section 5.2.2 below).

5.2 The NUM case

This section discusses exploration operators and related representational issues for NUM problems.

5.2.1 Representation

Starting from Section 2.1, this section treats NUM encoding in further detail; for illustration purposes, the molecular conformation problem is adopted again.

Real coding. The maximal relevant value range of a torsion angle runs from 0.0° to 360.0° due to periodicity. In general, the implementor will always use a finite range for each numerical problem parameter. (Note that explicitly using a finite value range for a problem parameter in a computer program is not restrictive, in principle, because in practice any variable occupies limited computer memory and therefore has a finite value range, implicitly.)

We will confine the discussion to real-coded (real-valued) numerical problem parameters, without loss of generality. For the i^{th} problem parameter, x_i, the value range can be defined in terms of a center value c_i and a deviation d_i:

$$x_i\text{'s value range} = [c_i - d_i\,,\, c_i + d_i) \qquad (1)$$

With the ranges of each problem parameter thus given, a continuous and finite Euclidean search space, or *search volume*, is defined. The center of a search volume is referred to as the *working point* (Figure 2). A search volume is called "successful" if it contains the true solution or an adequate solution.

In order to keep the number of candidate solutions that are contained in the search volume finite, a minimal step size (progression unit) is taken into account. This is usually accomplished by digitizing the range of each problem parameter into a fixed number of levels, leading to a discrete,

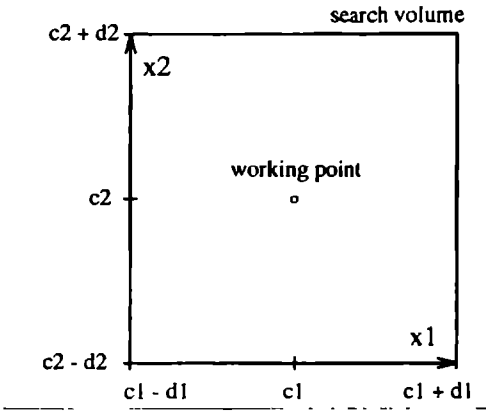

Figure 2: The center of a search volume is called the working point.

real coding. For convenience, the levels are taken equidistant. The number of equidistant levels on the range for a particular problem parameter is called the *encoding resolution*, or sometimes simply the *resolution*, of the problem parameter. In general, values for different problem parameters may be coded discretely with different resolutions. The resolution of problem parameter x_i is henceforth denoted as ρ_i. With all ρ_is specified, a discrete real search space, or *search grid*, is defined within the search volume (Figure 3). For prob-

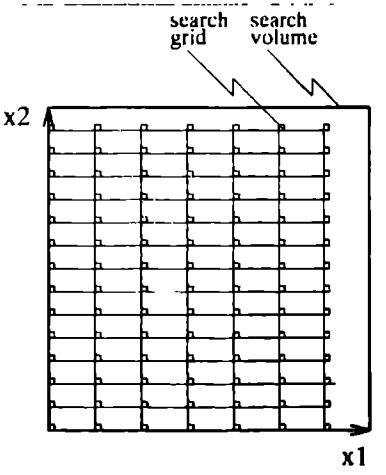

Figure 3: A two-dimensional search volume (continuous real-coded search space) and search grid (discrete real-coded search space).

lem parameter x_i, the distance between two adjacent levels on the search grid is called the mesh

size; the inverse mesh size – $\rho_i/(2d_i)$ – is called the *decoding resolution* of problem parameter x_i. The decoding resolutions indicate the search precision that can be attained in the real space.

The number of nodes on the search grid equals the formal size of the search space, and depends exponentially on the number of problem parameters. This property often leads to astronomical search spaces in practice. For instance, for 28 torsion angles and resolution 720 in a molecular conformation problem, the size of the search space is approximately 10^{80}; in many cases, this is prohibitively large for enumerative- or other naive search on today's or tomorrow's fastest computers.

The rationale for encoding. As alternatives to real coding, Section 2.1 introduced decimal and binary encoding. But why and when should one encode in the first place? After all, the encoded values (decimal or binary) need to be decoded to the real values that the objective function will take as arguments; that is, the need for decoding seems to entail an unwanted computational overhead, at first glance.

There are circumstances in which such an overhead is acceptable or even welcome, though. For instance, in many real-world applications, the computational overhead of decoding is negligible in comparison with the computational burden of string evaluation. More importantly, though, it is conceivable that only certain representations give access to recombination heuristics that result in optimal performance, i.e. in perception of the smoothest feasible o-landscape (Part 1 [56]). In this case, the true solution (or an adequate solution) may be found in less generations, possibly so much less that the overall running time is reduced despite the computational overhead of decoding prior to each string evaluation. Evidently, whenever such a net gain can be obtained, it makes sense to encode; in most cases, this can only be assessed empirically.

Practitioners who advocate binary encoding motivate their choice as follows. Firstly, more implementational convenience and greater speed of execution can be achieved, as computers naturally manipulate information in binary form. Secondly, a binary encoding leads to the highest ratio of Holland schemata to strings; see Part 1 [56] for more details on this theoretical argument. However, since such a maximal wealth of information is not necessarily relevant for the specific search task (Part 1 [56], [47, 85, 84]), non-binary encodings should be considered as possibly better choices in some cases.

Encoding terminology. The further discussion is facilitated by the following conventions and terminology concerning the encoding of real values for numerical problem parameters.

Recall from Part 1 [56] that the value of a numerical problem parameter can be encoded as a finite-length *word* composed of characters, or digits, copied from some k-cardinal alphabet ($k \geq 2$ and k is finite). A contiguous concatenation of words forms a *string* that encodes a candidate solution; in a D-dimensional problem, D words are concatenated. Exploration operators for NUM genetic algorithms fall apart into two classes: word-level operators (wherein the manipulated elements are words) and digit-level operators (wherein the manipulated elements are digits).

Different words in a string may have different lengths; the number of digits in the word for the i^{th} problem parameter, x_i, is denoted as ℓ_i; the resolution for x_i is $\rho_i = k^{\ell_i}$. The vector of word lengths for an arbitrary string in the population is called the *string format*: $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_D)$; all strings in a population have the same string format – e.g. $\boldsymbol{\ell} = (7, 2, 5)$ for a three-dimensional problem. In the special case that all word lengths are the same, the string format is referred to as "homogeneous" – e.g. $\boldsymbol{\ell} = (4, 4, 4)$. The sum of all word lengths is the string length, $L = \sum_{i=1}^{D} \ell_i$.

For $k = 2$ (binary encoding) a word would logically be called a bitword. In this specific case, however, the term *bitfield* is preferred in order to maintain the term "bitword" as used in computer terminology: a "natural", i.e. addressable, unit of computer memory in a particular hardware platform. A contiguous concatenation of bitfields forms a *bitstring*.

Abovementioned conventions are graphically exemplified in Figure 4 for binary encoding.

Decimal encoding. A decimal encoding ($k = 10$) decodes to the discrete real search space (the search grid) according to the following, linear digitization relation:

$$x_i = v_i + (w_i - v_i) \cdot \frac{D_i}{\rho_i} \qquad (2)$$

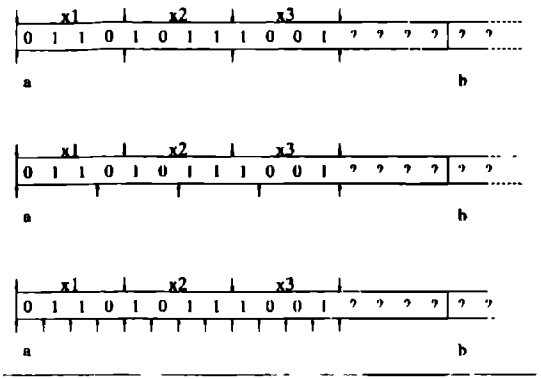

Figure 4: Top, middle, bottom: bitstring that encodes three values, for problem parameters x_1, x_2, x_3, respectively.
↓s delimit the bitfields that correspond with these values. The bitstring resides in a bitword (rectangle) with starting address "a" and has the homogeneous format $\ell = (4, 4, 4)$; ?-marks denote unused bits.
↑s delimit elements manipulated by an exploration operator. Bottom: elements are bits, resulting in bit-level exploration; larger elements may be viewed as virtual digits. Top: elements are bitfields (i.e. are "synchronous" virtual digits), resulting in bitfield-level exploration. Middle: elements are neither bits nor bitfields (i.e. are "asynchronous" virtual digits); this mode has not met widespread application.

where v_i and w_i are the real lowbound and -highbound, respectively, for the value range of problem parameter x_i, ρ_i is the encoding resolution, D_i ($< \rho_i$) is a decimal value that denotes the D_i^{th} level on the range. In terms of center value c_i and deviation d_i (Equation 1), we have $v_i = c_i - d_i$ and $w_i = c_i + d_i$, and Equation 2 becomes:

$$x_i = c_i + d_i \cdot \left(\frac{2D_i}{\rho_i} - 1 \right) \tag{3}$$

This decoding procedure is graphically depicted in Figure 5a for a candidate molecular conformation, using resolution $\rho_i = 8$ for the torsion angle values.

Binary encoding; the difference. Figure 5b illustrates the decoding of regular binary encoded values ($k = 2$). In order to better understand how the choice of the encoding can affect genetic

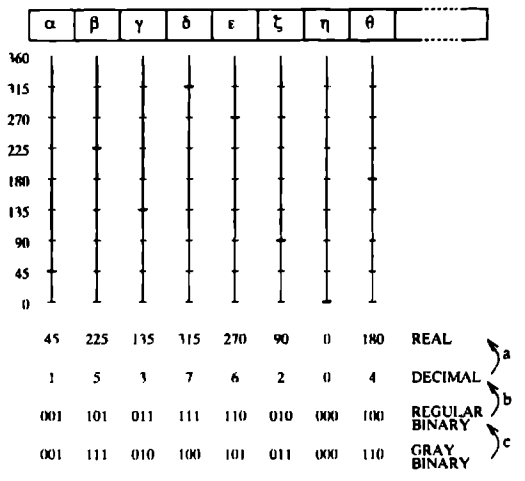

Figure 5: Summary of common decoding strategies in genetic algorithms for NUM problems, illustrated for a candidate molecular conformation defined by 8 torsion angles that take values in resolution 8.

search, consider two binary parent strings

$$\begin{aligned} {}^2P_1 &= 011\ 110\ 001\ 101 \\ {}^2P_2 &= 101\ 001\ 100\ 110 \end{aligned} \tag{4}$$

and two decimal parent strings

$$\begin{aligned} {}^{10}P_1 &= 3\ 6\ 1\ 5 \\ {}^{10}P_2 &= 5\ 1\ 4\ 6 \end{aligned} \tag{5}$$

Note that ${}^2P_i$ and ${}^{10}P_i$ ($i = 1$ or 2) have deliberately been chosen such that they represent the same candidate solutions in the real space (which is four-dimensional), i.e. ${}^2P_i$ decodes to ${}^{10}P_i$. Next, apply a digit-level recombination operator, say of type **1X**, to the binary parent strings ${}^2P_i$ and to the decimal parent strings ${}^{10}P_i$, respectively. From counting the number of possible **1X** recombinations – 11 and 3, respectively – it follows immediately that the binary-level **1X** has the largest exploratory power. In general, a digit-level recombination operator for a low-cardinality encoding has a higher exploratory power than a similar recombination operator for a high-cardinality encoding.

Two other recombination properties that are important to consider in connection with encodings are the recombinative potential and disrup-

tivity (see Section 5.1.1 above). Both properties depend on similarities between strings. In terms of Holland schemata, three similarities are inferred between ${}^2P_1$ and ${}^2P_2$ (by Holland schema **1 *** *0* 1**), whereas no similarities are inferred between ${}^{10}P_1$ and ${}^{10}P_2$ (by Holland schema * * * *). Hence, in the decimal encoding there is less similarity information available to recombine, and also less to disrupt. In general, a digit-level recombination operator for a low-cardinality encoding has a higher recombinative potential and is more disruptive than a similar recombination procedure for a high-cardinality encoding.

Incidentally, it is important to realize that the choice for binary encoding leaves open the possibility to simulate encodings with cardinality larger than 2. For instance, treating twosomes of adjacent bits as virtual digits, amounts to a quaternary encoding ($2^2 = 4$), threesomes (Figure 4, middle) to an octal encoding ($2^3 = 8$), etc. (That virtual digits share many properies with words should not strike one as surprising.) Moreover, adhering to one basic encoding in a universal way introduces a degree of uniformity, which is both convenient and efficient with regard to software development: all procedures for exploration in a general-purpose software library can make use of the same functional primitives for the modification of strings. On the other hand, however, one should realize that the use of a particular basic encoding, contributes to the overhead in string decoding. It depends on the application of interest whether this burden is significant in comparison with the burden related to string evaluation; in many real-world applications it appears to be insignificant.

Gray binary encoding. Numerous types of binary encoding are conceivable besides regular binary encoding. Practitioners have shown preference for so-called Gray binary encoding [11]. The key property of Gray binary encoding can be phrased as: for any pair of integers of which the absolute arithmetic difference equals 1, the respective Gray binary encodings differ at exactly one bit position. Decoding from Gray- to regular binary encoding – graphically illustrated in Figures 5c and 6 – comes down to a simple routine [68]; a version thereof in the computer language C (ANSI standard) is presented in [55].

The advantage of Gray binary encoding over regular binary encoding can be illustrated by the following example. In Figure 6, consider moving in the real-coded space from the level 135° to the next level, 180°, i.e. from 3 to 4 in the decimal-coded space. In the corresponding Gray binary

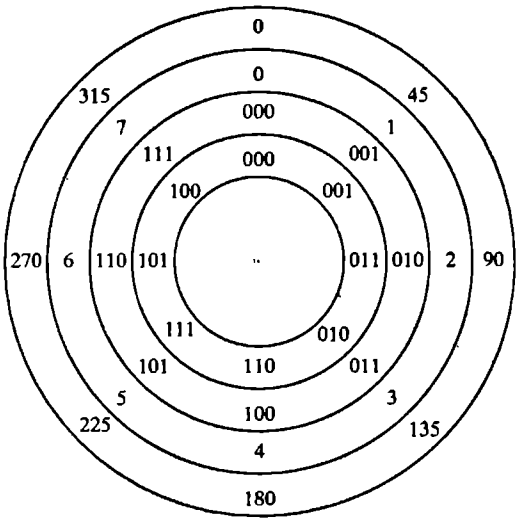

Figure 6: *Ibid* Figure 5, emphasizing periodicity. First (inmost) layer: Gray binary encoding; second layer: regular binary encoding; third layer: decimal encoding; fourth (outmost) layer: real coding.

space, this minimal step comes down to a move from 010 to 110 – separated by Hamming distance 1. (The Hamming distance between two equally sized bitfields is the number of positionwise differing bit values.) In the regular binary space, however, the move is from 011 to 100 – separated by Hamming distance 3.

Figure 7 illustrates the Hamming distance in regular- and Gray binary space as a function of minimal steps in decimal-coded space. It follows that if an o-landscape is "smooth" in a decimal space – that is, "smooth" in the real space, linearly related to it – then in regular binary space it appears more "rugged" (due to the Hamming gaps) than it is in Gray binary space. Accordingly, exploration operators can perceive a lower apparent problem complexity in the Gray binary space, resulting in better search performance. These theoretical arguments are empirically corroborated by numerous comparative investigations reported in the literature. In passing, it should be noted

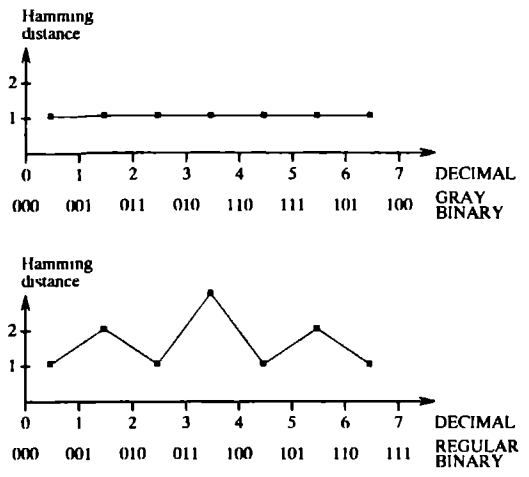

Figure 7: Hamming distance in binary space as a function of minimal steps in decimal-coded space. Bottom: regular binary space. Top: Gray binary space.

that Gray binary encoding basically amounts to a fix for artifacts (Hamming gaps) that would arise from representing real values in regular binary encoding.

An objection sometimes raised against Gray binary encoding is that it is cyclic. For instance, the lowest and highest decimal values in Figure 6 are 0 and 7, respectively, but in Gray binary space these bounds are separated by the minimal nonzero Hamming distance, 1; in regular binary space, in contrast, the Hamming distance is maximal, 3, which seems more consistent with the decimal encoding. Indeed, only in some applications – namely those that involve periodic problem parameters (such as the torsion angles in aforementioned molecular conformation problem) – advantage can be taken of the cyclic property of Gray binary encoding.

5.2.2 Recombination operators

The recombination operators discussed in this section – **B_MX**, **B_MX'**, **B_UX** and **B_UX'** – are most commonly used in NUM problem solving. They are applied to bitstrings and fall apart into bit-level operators and bitfield-level operators; the latter are distinguished by a prime (') appended to their name. As will be clarified below, the operators can be described in terms of random binary templates as defined in Section 5.1.3 above. They belong to the class of positionwise swapping recombination operators: the elements in each pair of swapping elements in the parent strings undergoing the recombination are at equal positions, i.e. they are uniquely indicated by exactly one 1-bit in the binary template. (Non-positionwise swapping recombination operators are common in, inter alia, SUB-, SEQ- and other combinatorial problem solving; see Sections 5.4.2 and 5.5.1 below.)

B_MX and B_MX'. The salient properties of the **B_MX** operator, which is described in [39, 17], were recently outlined in [21, 75]. The operator is applied with probability p_r, to two parent strings with length L (in bits). It can be described in terms of a random L-bit $T_M(n_b)$ binary template, where the control parameter n_b is the number of effectuated breakpoints ($1 \leq n_b < L$): the 1-bits in the template select the pairs of swapping bits in the parent strings; the operation is graphically summarized in Figure 8 for $L = 10$ and $n_b = 2$.

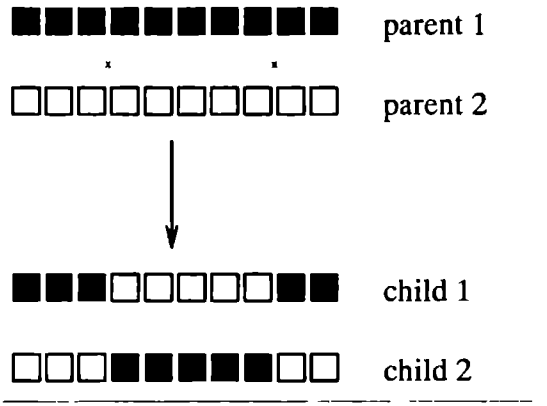

Figure 8: **B_MX** recombination ($L = 10$ and $n_b = 2$).

For $n_b = 1$ one obtains **B_1X** (which in Part 1 [56] and elsewhere in this tutorial part is denoted as **1X** – simple recombination); for $n_b = 2, 3$, etc., one obtains **B_2X**, **B_3X**, etc. Among these, **B_1X** and **B_2X** have traditionally enjoyed most frequent usage – based on the classical conviction that little disruption of Holland schemata leads to the good performance under all circumstances. In this light, the use of **B_1X** is motivated by the argument that disruptivity decreases

when n_b decreases. **B_2X** reflects a compromise between striving for minimal n_b and the theoretical argument that no disruption can occur for an even number of breakpoints if these fall between each of the defining positions of a Holland schema [17, 75]. Indeed, in many applications it has been observed that **B_2X** outperforms **B_1X**. To date, a more matured view on disruptivity prevails: high disruptivity is not considered a disadvantage *per se*, provided it is properly counterbalanced by selection pressure, as pointed out above (see Section 5.1.1).

The **B MX'** operator is similar to **B MX**, except that the swapped string elements are, of course, bitfields instead of bits. The operator is applied with probability p_r to two parent strings with length D (in bitfields), using a random D-bit $T_M(n_b)$ binary template ($1 \leq n_b < D$); that is, the choice of the breakpoints is constrained such that these always fall between adjacent bitfields.

B UX and B UX'. The salient properties of the **B UX** operator, which is described in [80], were recently outlined by [21, 76]. (The idea behind this operator goes back in time at least to Reed *et al.* [66].) The operator is applied with probability p_r to two parent strings with length L (in bits). It can be described in terms of a random L-bit $T_U(n_s)$ binary template, where the control parameter n_s is the number of effectuated swaps ($1 \leq n_s < L$): the 1-bits in the template select the pairs of swapping bits in the parent strings; the operation is graphically summarized in Figure 9 for $L = 10$ and $n_s = 5$.

The ratio n_s/L is called the bit swap rate, r_s. It is easily verified that when the role of selected- and unselected bit positions is interchanged, the same child strings are obtained. This symmetric property of **B UX** implies that $n_s = x$ and $n_s = L - x$, i.e. $r_s = y$ and $r_s = 1.0 - y$, are equivalent parameterizations; hence, maximal exploration of the search space is attained for $r_s = 0.5$. Incidentally, **B UX** may be implemented alternatively such that r_s is distributed. Then, r_s is derived from p_s: the probability by which each of the L possible positionwise bit swaps between the strings actually occurs; r_s is binomially distributed, with mean p_s and variance $p_s(1 - p_s)$. For this implementation, **B UX** can be described in terms of a random L-bit $T_U(p_s)$ binary template.

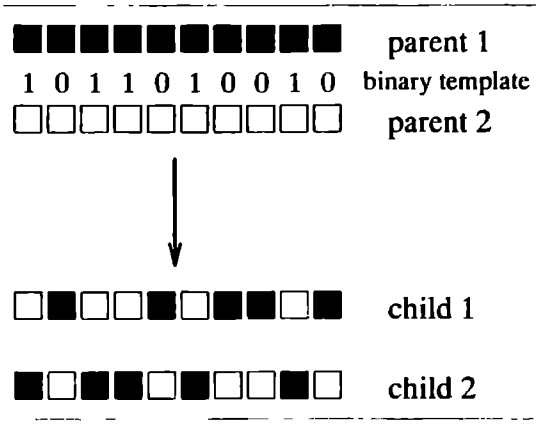

Figure 9: **B_UX** recombination ($L = 10$ and $n_s = 5$).

The **B_UX'** operator is similar to **B UX**, except that the swapped string elements are, of course, bitfields instead of bits. The operator is applied with probability p_r to two parent strings with length D (in bitfields), using a D-bit $T_U(n_s)$ binary template ($1 \leq n_s < D$).

Positional bias and exploratory power. That **B MX**, **B_MX'**, **B UX** and **B UX'** have found widespread use in genetic practice is remarkable insomuch as these operators do not heuristically incorporate the building block principle: that is, the recombination heuristics of the operators are comparatively simple – not aimed at intelligently detecting and preserving predefined properties (schemata) that are stored in strings and that are important as they contribute to fitness.

In **B MX** and **B MX'**, the building block principle can, however, be implemented through the placement of the bitfields in the bitstrings. In particular, practitioners tend to place the bitfields for any given pair of problem parameters adjacently in the bitstrings if it is known that between them there is strong epistasis (i.e. correlation in the sense that large fitness contributions occur only when both parameters adopt certain values simultaneously), because this maximizes the chance that both bitfields are propagated together from a parent string to a child string – hence that important information is preserved. If this positional bias of **B MX** and **B MX'** – i.e. (Part 1 [56], [21]) if the effect of bitfield placement on the per-

formance of these operators – is taken advantage of as indicated above, then the building block principle applies.

For certain target problems, however, it is difficult or impossible to exploit positional bias. For instance, it may be the case that unknown correlations exist between the problem parameters. Then, the number of bitfield orders that one would have to try in order to find the optimal order, is in many cases prohibitively large (e.g. approximately 10^{158} for a 100-dimensional target problem). Moreover, the proportion of correlated problem parameters may be so large that it is simply impossible to place the bitfields for each correlated pair closeby in a string. Under these circumstances, practitioners will normally seek ways to apply less positionally biased recombination, since not exploiting positional bias present in recombination often works out detrimental (i.e. leads to deception).

The positional bias of **B MX** and **B MX'** can be decreased – but not eliminated – by increasing n_b [21]. It is important to realize that thereby the exploratory power increases, which is intuitively plausible. Of course, this change in the exploratory power of recombination generally gives rise to the need for a change in the selection pressure such as to restore the balance between exploration and exploitation.

The exploratory power of **B MX'** for a particular n_b is smaller than that of **B MX** for the same n_b, because in the latter case smaller string elements (i.e. bits) are manipulated. Similarly, the exploratory power of **B UX'** for a particular n_s is smaller than that of **B UX** for the same n_s. In general, the comparatively small exploratory power of bitfield-level recombination is attractive when the correct bitfields are already present in the population but not yet combined in one string. In order to achieve these circumstances, bitfield-level recombination leans heavily on mutation (discussed in the next section).

Maximal exploratory power and minimal positional bias of **B MX** is achieved for $n_b = L - 1$. Maximal exploratory power of **B UX** is achieved for $n_s = 0.5$ and is larger than the exploratory power of **B MX** [76]; this can be verified by a simple example: when one parent string comprises all 1s and the other all 0s, then the child set for **B UX** is the entire search space, whereas for **B MX** the child set is always merely a particular subspace of the search space. Importantly, the positional bias of **B UX** is zero for any n_s. These properties of **B UX** – zero positional bias and wide adjustability of the exploratory power – make the operator an attractive choice when complex target problems are dealt with, i.e. problems with a large proportion of strongly correlated problems parameters; according to our empirical findings, a bitswap rate $r_s \approx 0.3$ is normally a good choice.

Variations on the theme. The designer may apply any customizations or modifications to a particular recombination procedure which he/she believes will improve performance.

Averaging recombination. The basic recombinative interaction in most recombination operators is element swapping between parent strings. However, NUM problem solving is special in that it allows basic recombinative interactions of an arithmetic nature as well. For instance, an intuitively appealing kind of basic recombinative interaction might be to numerically average, instead of swap, bitfields between parent strings [15, 45]. Note that any averaging recombination operator produces only one child string per pair of parent strings.

Productivity enhancement by constraints. The selection of pairs of swapping elements in a pair of parent strings by the recombination operator can incidentally proceed such that each pair is homogeneous (i.e. contains identical elements). Evidently, when that is the case, both child strings produced are copies of their respective parents. In order to combat unproductive recombination, constraints need to be imposed which strive after at least one heterogeneous pair of swapping elements; in doing so, a more intelligent recombination operator is obtained.

A simple approach towards productive recombination is obtained by incorporating an unconstrained recombination operator into a *rejection loop*. For any proposed pair of parent strings, the loop is repeated as long as unproductive recombinations would occur; these trials are rejected. The first productive recombination is accepted. An advantage of rejection strategies is that they are universally applicable. A disadvantage is that they may be too time consuming: as the parent strings

become increasingly similar in the course of the evolution (due to the decay in population diversity), the average number of rejections increases. In the worst case, a deadlock occurs, so that it is necessary to impose a maximal number of trials beyond which an unproductive recombination is accepted as inevitable.

A more advanced approach towards productive recombination is obtained by modifying an unconstrained recombination operator, taking advantage of foreknowledge about the latter. For instance, if a positionwise swapping recombination procedure is used (e.g. **B_MX** or **B_UX**), one knowns in advance that the elements in each pair of swapping elements are at equal positions, by definition. Therefore, it is particularly easy to locate what would become homogeneous pairs of swapping elements and to exclude these from participation. A parent string which recombines in such a constrained way is called a *reduced surrogate* [7]. Only when "void" reduced surrogates come up, unproductive recombination is accepted as inevitable.

The use of reduced surrogates does not only maximize the productivity of recombination. There is an additional, more subtle advantage, phrased by Whitley as follows [87]: "By looking at reduced surrogates one is also heuristically reducing the search space. In other words, the two parents are in agreement about the values of certain bit positions. This agreement has the effect of reducing that part of the search space which is currenly of interest, especially if one looks at the amount of agreement that exists across several recombinations. The reduced surrogate operator reflects that agreement. This ensures that offspring are selected from a part of the search space about which the parents are not in agreement."

5.2.3 Mutation operators

The mutation operators discussed in this section – **B M, B_M'** – are most commonly used in NUM problem solving. They are applied to bitstrings and fall apart into bit-level operators and bitfield-level operators; the latter are distinguished by a prime (') appended to their name.

The **B_M** operator – bit-level jump mutation, or simple mutation – is applied to one string with length L (in bits), and can be described in terms of a random L-bit $T_U(p_m)$ binary template, where p_m is the mutation probability: the 1-bits in the template select the flipping bits in the target string.

The **B M'** operator can be applied in two modes: bitfield-level jump mutation and bitfield-level step mutation. In both cases, it is applied to one string with length D (in bitfields), and can be described in terms of a random D-bit $T_U(p_m)$ binary template, where, again, p_m is the mutation probability: the 1-bits in the template select bitfields in the target string that are either "jumped" or "conditionally stepped", depending on the mode. For selected bitfield i with length ℓ_i (in bits), both cases are discussed next (Figure 10).

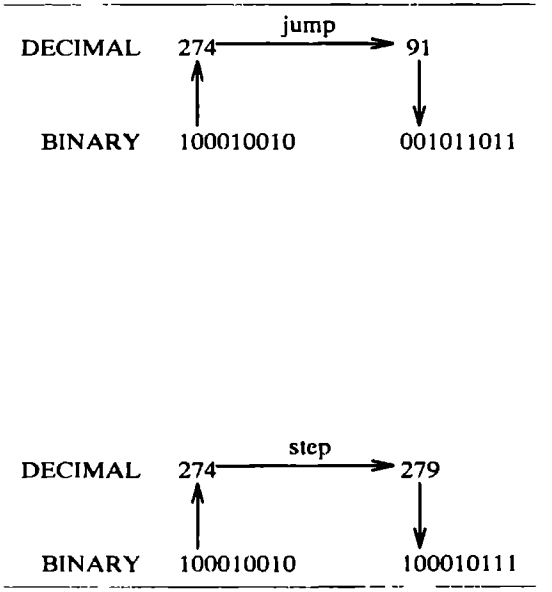

Figure 10: **B_M'** mutation of a 9-bit bitfield. Top: jumping mode. Bottom: stepping mode (step size 5).

A bitfield-level jump mutation is performed as follows: the bitfield is decoded to a decimal value, which is then modified to a random value within the range $[0, 2^{\ell_i})$, and finally the bitfield is updated accordingly.

A bitfield-level step mutation [55] is performed as follows: the bitfield is decoded to a decimal value, which is then either incremented (with probability 0.5) or decremented (otherwise) by a fixed integer step, subject to the condition that the decimal value is not changed if it would either overflow (i.e. become larger than $2^{\ell_i} - 1$) or underflow (i.e. become smaller than 0); finally, the

bitfield is updated accordingly.

Variations on the theme. The designer may apply any customizations or modifications to a particular mutation operator which he/she believes will improve performance. As an example, we consider probabilistic mutation next.

First, note that mutation by step-mode **B M'** is random in that all bitfields in the target strings have an equal chance (p_m) to be stepped. This also implies, however, that single bits in each bitfield have different chances to be flipped: the more significant bits are flipped by a lower chance (thus indeed suppressing large jumps). So, in this sense bitfield-level step mutation is probabilistic.

A similar principle can easily be implemented in **B_M** by modifying the latter in such a way that p_m decreases with the significance of the bits in each bitfield; then, large changes in decoded values are less likely than small changes. — In a comparative study involving genetic algorithms wherein mutation was assigned an important role, Fogarty [26] and Bramlette [8] observed significant improvements in performance with **B_M** modified such that p_m decreases exponentially with the significance of the bits in each bitfield.

5.3 General notes on exploration operators for SUB- and SEQ problems

The following subsections serve to facilitate the forthcoming description of exploration operators for SUB- and SEQ problems.

5.3.1 Dealing with combinatorial encoding constraints

In this section, we outline the main strategies that practitioners have employed up to now in dealing with combinatorial encoding constraints related to permutation encoding for SEQ problems and to direct subset encoding for fixed-size SUB problems (see Sections 2.2 and 2.3 above), and highlight the most popular among these. (Exploration operators for unknown-size SUB problems and their underlying binary subset encoding are hereafter discarded.)

In Section 4.6 above, we showed that simple recombination, **1X**, applied to two permutations, may produce two non-permutations, thus violating permutation encoding constraints. This argument applies generally to any recombination operator in the **MX** and **UX** families, and to any mutation operator in the **M** family (simple mutation). In a similar way, these operators are easily seen to be incompatible with direct subset encoding constraints as well.

The strategies that have been contrived to deal with the constraints inherent to both types of combinatorial encoding, fall apart into two basic categories: *corrective* constraint handling and *preventive* constraint handling.

Corrective constraint handling. In corrective handling of encoding constraints, any known exploration operator is, in principle, allowed – even when the operator is incompatible with present encoding constraints. First, the exploration operator is naively applied, implying that illegal strings may be produced. When such violations occur, corrective action can be undertaken according to one of the following strategies:

1. The illegal strings are *lethalized* by assigning them a zero raw fitness value. In this way, the rate at which incidentally produced illegal strings are reproduced becomes zero;

2. The illegal strings are *penalized* by assigning them a comparatively low raw fitness value. In this way, the rate at which incidentally produced illegal strings are reproduced can become low enough so that the search will not be frustrated by a large proportion of illegal strings in the population, while the few illegal strings present may provide sensible avenues to escape local optima;

3. The illegal strings are *repaired* ("patched") according to some predefined criterion such that they become legal.

In passing, note that the corrective strategies listed above all have biological counterparts: for, many natural life forms have the capability to repair incidental flaws in their DNA; without repairment taking place, the individuals soon become weak (i.e. are penalized) or even die (i.e. are lethalized).

In practice, the rate at which illegal strings are incidentally produced and are subsequently subjected to corrective action, depends on the chosen exploration operator and on the nature of the target problem. Evidently, a higher rate of corrective

action amounts to a larger waste of computational efforts, thus making the search less efficient.

An example of the use of an exploration operator that is incompatible with encoding constraints but that is still acceptable in practice owing to a low rate of corrective action required, is discussed in [52, 48], concerning an application of genetic algorithms in wavelength selection (as an unknown-size subset selection problem). In this application, a 36-bit binary encoding is used, subject to the constraint that the number of 1-bits in any bitstring may not be smaller than 4. Despite the fact that the exploration operators used in this application (among which **2X**) are incompatible with this encoding constraint, it turns out that the rate by which illegal bitstrings are incidentally produced is extremely small. This may be appreciated intuitively by considering the probability by which the number of 1-bits that occur among 36 random bit values is less than four: $7807/2^{36} \approx 1.1 \cdot 10^{-7}$. (Of course, this is merely a rough estimate, because genetic sampling is not random but probabilistic.) On the rare occasions that an illegal string is incidentally produced, it is simply lethalized.

Preventive constraint handling. In preventive encoding constraint handling, the heuristics of a modification (exploration) operator and/or its underlying encoding are designed or chosen carefully such that it is guaranteed that illegal strings are never incidentally produced. Preventive action can be undertaken according to one of the following strategies:

1. An unconstrained, usually complex, sub-encoding for the constrained encoding is used;
2. Sophisticated – i.e. intelligent, complex – modification heuristics are used.

As an example of the first preventive strategy, consider a heuristically simple recombination operator such as **2X**. Although this operator is not compatible with, say, permutation encoding, as pointed out above, it can nevertheless be used, in principle, provided that an unconstrained sub-encoding for the permutation encoding exists. In this way, when the operator is applied to strings according to the sub-encoding, the strings thereby produced will, after decoding, always be legal according to the permutation encoding. That this strategy, nevertheless, has met only sporadic application, may be ascribed to the fact that [16]: (1) a complex sub-encoding is unfriendly to the designer's intuition used in implementing the building block principle; (2) a complex sub-encoding tends to roughen up the o-landscape, or at least change its shape in an inscrutable way.

In the second preventive strategy, the modification heuristics are designed in a special, intelligent way such that the strings will always be legal according to the encoding used. Apparently having gained most popularity, this approach is elaborated further on below. In passing, one may note that both strategies for preventive encoding constraint handling do not seem to be grounded in biology. Such aberrations, however, hardly preoccupy the practitioner when the purpose of problem solving is served, as was pointed out in Part 1 [56].

Among modification operators based on the second preventive strategy, recombination operators turn out to be heuristically more sophisticated than mutation operators, as a rule. This is not too surprising, because here recombination must not only comply with the encoding constraints, but also, simultaneously, accomplish the relevant preservation task (see next section).

5.3.2 Principal preservational properties

Principal preservational properties for SUB problems, using direct subset encoding, can be summarized as [53]:

- **Identity**. A child string inherits elements in such a way that the identity of elements in both parent strings is reflected as much as possible. Identity is the trivial and most fundamental preservational property, since the other preservational properties (listed immediately below) are undefined when the element identities are unknown.
  Example: if one of the parent strings is 6 4 8 1 2 9 and the child string turns out to be 1 5 0 3 4 7, then the identity of elements 1 and 4 has been preserved.
- **Position**. A child string inherits elements in such a way that the position of elements in both parent strings is reflected as much as possible.
  Example: if one of the parent strings is 6 4 8 1 2 9 and the child string turns out

to be 2 4 7 1 3 8, then the position of elements 1 and 4 has been preserved.

- **Order.** A child string inherits elements in such a way that the order of elements in both parent strings is reflected as much as possible.
  Example: if one of the parent strings is 6 4 8 1 2 9 and the child string turns out to be 2 8 4 7 0 1, then the order of elements 1 and 4 has been preserved.

- **Adjacency.** A child string inherits elements in such a way that the mutual adjacency of elements in both parent strings is reflected as much as possible.
  Example: if one of the parent strings is 6 4 1 2 8 9 and the child string turns out to be 5 9 8 1 4 2, then the adjacency of elements 1 and 4 has been preserved.

It is important to realize that these properties are not mutually independent. For instance, when the position of all elements in a string is known, then their order and adjacency is also known. It depends on the purposes of the search task exactly which property (or properties) should be preserved in order to fulfil the building block principle.

Except for element identity, the principal preservational properties for SEQ problems, using permutation encoding, are the same as those for SUB problems. (That element identity is an irrelevant preservational property for SEQ problems, follows immediately from the fact that all possible candidate permutations for any SEQ problem contain the same elements, by definition.)

The following sections discuss exploration operators for genetic algorithms applied to SUB- and SEQ problems; the operators manipulate single decimal digits.

5.4 The SUB case

The exploration operators discussed in this section are based on direct subset encoding (Section 2.2); this encoding is elaborated upon some more next.

5.4.1 Representation

Direct subset encoding can be accomplished as follows [53]. A candidate subset is represented as the leftmost l elements in a permutation of L ($> l$) elements; the latter comprise the source set. The rightmost $L - l$ elements represent the so-called complementary subset.

5.4.2 Recombination operators

In this section, the recombination operators **D SX** and **D MX** are briefly passed in review; among these, **D MX** is described in detail.

D SX – general-purpose subset recombination – was developed by Lucasius and Kateman [53]. The operator can be used in three principal modes: for the preservation of element identity, -position and -order information, respectively, in the contribution from the parent strings. When only identity information is important for the search task, we speak of a regular SUB problem; otherwise, the problem is called a sequenced SUB problem [53, 54, 55]. Incidentally, note that compliance with encoding constraints is a sufficient condition to preserve identity; further sophistication of the recombination heuristics is required when additional properties need to be preserved.

D MX – mix subset recombination – distinguishes itself from **D SX** in that it is confined to regular SUB problems and applies a built-in mutation operator.

Description of D MX. For illustration purposes, consider the next parent strings of length $l = 3$:

$$P_1 = 2\ 3\ 7$$
$$P_2 = 4\ 8\ 2$$

The production of the child strings, C_1 and C_2, by **D MX** can be described as follows:

1. Mix P_1 and P_2:

   (a) Append copies of P_1 and P_2 to obtain, say, Q:

   $$Q = 2_1 3_1 7_1 4_2 8_2 2_2$$

   The subscripts are not actually part of the elements, but merely serve to remind of the original parent.

   (b) Randomly scramble the elements, e.g.:

   $$Q = 4_2 2_2 2_1 8_2 7_1 3_1$$

2. Add new material; that is, apply the following built-in mix mutation:
   With a predetermined probability $p_{m,\mathrm{mix}}$, replace each of the first l consecutive elements in Q by a copy of an element indicated randomly, but never more than once, in the source set, e.g.:

   $$Q = 5\ 2_2 7\ 8_2 7_1 3_1$$

   Note that in this example, 2 of the l trials succeeded (at the first and the third position).

3. Randomly scramble the elements again, e.g.:

   $$Q = 2_2 7_1 7\ 3_1 5\ 8_2$$

4. Build C_1 by copying l elements from Q, starting at the leftmost element and going elementwise to the right, subject to the condition that elements that are already in C_1 are skipped:

   $$C_1 = 2\ 7\ 3$$

5. Build C_2 by copying l elements from Q, starting at the rightmost element and going elementwise to the left, subject to the condition that elements that are already in C_2 are skipped:

   $$C_2 = 8\ 5\ 3$$

5.4.3 Mutation operators

Important mutation operators for SUB problems are **D_RM** and **D_TM** – reorder mutation and trade mutation, respectively; for their description, the reader is referred to [53, 55].

5.5 The SEQ case

The exploration operators discussed in this section are based upon permutation encoding (Section 2.3); this encoding is not elaborated further upon here.

5.5.1 Recombination operators

In this section, the recombination operators **P_PMX**, **P_CX**, **P_OX1**, **P_OX2** and **P_EX** are briefly passed in review; among these, **P_PMX**, **P_CX** and **P_OX2** are described in detail.

P_PMX – partially matched recombination – was developed by Goldberg and Lingle [33, 29, 77, 27]. In a child string produced by this operator, the element position, -order and -adjacency information in the contribution from one of the parents is preserved.

P_CX – cycle recombination – was developed by Oliver *et al.* [64, 77, 27]. In a child string produced by this operator, the element position, -order and -adjacency information in the contribution from one of the parent strings is preserved; the element position information in the contribution from the other parent string is also preserved.

P_OX1 – order-based two-point recombination – was developed by Davis [12, 77, 27]. In a child string produced by this operator, the element position, -order and -adjacency information in the contribution from one of the parents is preserved; the element order information in the contribution from the other parent is also preserved.

P_OX2 – order-based uniform recombination – was developed by Syswerda [81, 15, 77]. In a child string produced by this operator, the element position, -order and -adjacency information in the contribution from one of the parents is preserved; the element order information in the contribution from the other parent is also preserved.

P_EX – edge recombination – was developed by Whitley *et al.* [91, 92, 77, 27]. In a child string produced by this operator, the element adjacency information contained in the overall contribution from both parent strings is optimally preserved. An enhanced version of the operator is described in [77]

Description of P_PMX. For illustration purposes, consider the next parent strings of length $L = 10$:

$$P_1 = 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$
$$P_2 = 2\ 6\ 4\ 1\ 9\ 8\ 3\ 0\ 5\ 7$$

The production of the first child string, C, by **P_PMX** can be described as follows:

1. Create a random $T_M(2)$ binary template of a size that matches the size of a parent string;

for example:

$$T_M(2) = 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0$$

2. Let C initially be a copy of P_1:

$$C = 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$

3. To facilitate visual verification of the operations to be performed next, consider C, $T_M(2)$, and P_2 aligned:

$$\begin{array}{lll} C & = & 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5 \\ T_M(2) & = & 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ P_2 & = & 2\ 6\ 4\ 1\ 9\ 8\ 3\ 0\ 5\ 7 \end{array}$$

Iterate from left-to-right for the 1-bits in $T_M(2)$, and let i thereby indicate the consecutive bit positions:

(a) Select in C the element at position i; in the first iteration:

$$C = 1\ 3\ \underline{8}\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$

(b) Select in C the element identical to the element in P_2 at position i; in the first iteration:

$$C = 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$

(c) Swap both selected elements in C; in the first iteration:

$$C = 1\ 3\ \underline{4}\ 8\ 7\ 0\ 2\ 9\ 6\ 5$$

(d) If iteration is not yet completed, go to (a).

It can be easily verified that the consecutive iterations in the present example yield the following Cs:

$$\begin{array}{lll} C & = & 1\ 3\ 4\ 8\ 7\ 0\ 2\ 9\ 6\ 5 \\ C & = & 8\ 3\ 4\ 1\ 7\ 0\ 2\ 9\ 6\ 5 \\ C & = & 8\ 3\ 4\ 1\ 9\ 0\ 2\ 7\ 6\ 5 \\ C & = & 0\ 3\ 4\ 1\ 9\ 8\ 2\ 7\ 6\ 5 \\ C & = & 0\ 2\ 4\ 1\ 9\ 8\ 3\ 7\ 6\ 5 \end{array}$$

The last C represents the first child string produced by **P PMX**. In order to create the second child string. the procedure is repeated starting from Step 2 with the roles of the parent strings interchanged.

Description of P CX. For illustration purposes, consider the next parent strings of length $L = 10$:

$$\begin{array}{lll} P_1 & = & 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5 \\ P_2 & = & 2\ 9\ 4\ 1\ 7\ 0\ 8\ 3\ 5\ 6 \end{array}$$

The production of the first child string, C, by **P_CX** can be described as follows:

1. Create a random $T_U(1)$ binary template of a size that matches the size of a parent string; for example:

$$T_U(1) = 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0$$

2. Let C initially be a copy of one of the parent strings, selected randomly; for example if P_1 was selected:

$$C = 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$

3. Let i indicate the bit position of the 1-bit in $T_U(1)$. Further, let j indicate a bit position in C, and let the initial value of j be the value of i.

4. To facilitate visual verification of the operations to be performed next, consider C, $T_U(1)$, and P_2 aligned:

$$\begin{array}{lll} C & = & 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5 \\ T_U(1) & = & 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ P_2 & = & 2\ 9\ 4\ 1\ 7\ 0\ 8\ 3\ 5\ 6 \end{array}$$

Iterate as follows:

(a) Select in C the element at position j; in the first iteration:

$$C = 1\ 3\ 8\ \underline{4}\ 7\ 0\ 2\ 9\ 6\ 5$$

(b) Select in P_2 the element identical to the element selected in C; in the first iteration:

$$P_2 = 2\ 9\ \underline{4}\ 1\ 7\ 0\ 8\ 3\ 5\ 6$$

(c) Replace the element selected in C by the j^{th} element in P_2; in the first iteration:

$$C = 1\ 3\ 8\ \underline{1}\ 7\ 0\ 2\ 9\ 6\ 5$$

(d) Set j to the position of the element selected in P_2.

(e) If j equals i, terminate iteration. Otherwise, go to (a).

It can be easily verified that the consecutive iterations in the present example yield the following Cs :

$$\begin{array}{lllllllllll} C & = & 1 & 3 & 8 & 1 & 7 & 0 & 2 & 9 & 6 & 5 \\ C & = & 1 & 3 & 4 & 1 & 7 & 0 & 2 & 9 & 6 & 5 \\ C & = & 1 & 3 & 4 & 1 & 7 & 0 & 8 & 9 & 6 & 5 \\ C & = & 2 & 3 & 4 & 1 & 7 & 0 & 8 & 9 & 6 & 5 \end{array}$$

The last C represents the first child string produced by **P CX**. In order to create the second child string, the procedure is repeated starting from Step 2 with the roles of the parent strings interchanged.

Note that as each child string produced by **P_CX** inherits, positionwise, an element either from P_1 or P_2; there is no positional disruption at all, i.e. element position is optimally preserved. While this seems good for search tasks wherein element position information must be preserved, it should be remarked that **P CX** has a comparatively strong tendency to produce child strings that are identical to their parents, i.e. to be unproductive.

Description of P OX2. For illustration purposes, consider the next parent strings of length $L = 10$:

$$\begin{array}{lllllllllll} P_1 & = & 1 & 3 & 8 & 4 & 7 & 0 & 2 & 9 & 6 & 5 \\ P_2 & = & 5 & 2 & 9 & 4 & 8 & 3 & 6 & 1 & 7 & 0 \end{array}$$

The production of the first child string, C, by **P OX2** can be described as follows:

1. Create a random $T_U(0.5)$ binary template of a size that matches the size of a parent string; for example:

$$T_U(0.5) = 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1$$

2. Let C initially be a copy of P_1:

$$C = 1\ 3\ 8\ 4\ 7\ 0\ 2\ 9\ 6\ 5$$

3. To facilitate visual verification of the operations to be performed next, consider C, $T_M(0.5)$, and P_2 aligned:

$$\begin{array}{lllllllllll} C & = & 1 & 3 & 8 & 4 & 7 & 0 & 2 & 9 & 6 & 5 \\ T_U(0.5) & = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ P_2 & = & 5 & 2 & 9 & 4 & 8 & 3 & 6 & 1 & 7 & 0 \end{array}$$

4. Select in C the elements that positionwise correspond with the 1-bits in $T_U(0.5)$:

$$C = 1\ \underline{3}\ \underline{8}\ 4\ 7\ 0\ 2\ 9\ 6\ \underline{5}$$

5. Select in P_2 the elements identical to the elements selected in C:

$$P_2 = 5\ 2\ 9\ 4\ \underline{8}\ \underline{3}\ 6\ 1\ 7\ 0$$

6. Rearrange the elements selected in C according to the order of the elements selected in P_2:

$$C = 1\ \underline{5}\ \underline{8}\ 4\ \underline{3}\ 0\ 2\ 9\ 6\ \underline{7}$$

This C represents the first child string produced by **P OX2**. In order to create the second child string, the procedure is repeated starting from Step 2 with the roles of the parent strings interchanged. Instead of using an element swap rate of 0.5 (as implied by the use of a random $T_U(0.5)$ binary template), **P OX2** may be generalized such that any element swap rate between 0.0 and 1.0 is allowed.

5.5.2 Mutation operators

An important mutation operator for SEQ problems is **P_RM**, which is identical to **D_RM** – reorder mutation (see Section 5.4.3 above).

6 Exploitation

This section passes in review a number of commonly used exploitation strategies.

6.1 Fitness scaling

In a genetic algorithm, either the reproduction procedure or the replacement procedure, or both, use a vector of fitness values or -ranks as input. Recall that a vector of raw fitnesses is obtained when all strings in the population are evaluated by the objective function. We denote this vector as:

$$\mathbf{f}_0 = (f_{0,1}, \cdots, f_{0,N})$$

where the subscript 0 reminds of the fact that the fitnesses are raw (i.e. not scaled); N is the number of strings in the population ($N \geq 2$, otherwise recombination can not be applied) - a control parameter.

When $\mathbf{f}_0$ is not suitable for use by the reproduction- and/or replacement procedure, it must be scaled. The need for fitness scaling arises, for instance, when the fitness vector is not normalized, while the reproduction procedure demands that it is. In another example, when in the fitness vector the maximal fitness is much larger than the average fitness, and strings are reproduced at rates proportional to their fitness (i.e. by means of roulette selection, discussed below), fitness scaling is needed to control the level of competition among members in the population such as to oppose premature convergence. Important work in this field has been done by Baker [1, 2, 3].

In practice, fitness scaling is usually conducted in a sequence of steps involving different, elementary fitness scaling functions. By chosing a number of elementary fitness scaling functions from a standard collection, and then applying them in some sequence, a wide variety of more complex fitness scaling functions can be assembled.

6.1.1 Translation and description of other scaling steps

The first fitness scaling step is oftentimes translation, which means that the transfer function, ξ_1, describing this step is defined by the relation:

$$f_{1,i} = \xi_1(f_{0,i}) = f_{0,i} - (f_{0,min} - \varepsilon)$$

where $f_{0,min}$ is the smallest fitness in $\mathbf{f}_0$; ε is a control parameter ($\varepsilon \geq 0.0$), called "fitness offset". The purpose of translation is to affect fitness ratios, or *competition*, between strings; competition is maximal for $\varepsilon = 0.0$.

In general, the transfer function for the n^{th} fitness scaling step ($n \geq 1$) can be formally summarized as:

$$\mathbf{f}_n = \xi_n(\mathbf{f}_{n-1})$$

A fitness scaling function, ξ_n, makes practical sense only if it fulfils the following monotonicity condition: if $f_{n-1,i} \leq f_{n-1,j}$ then $f_{n,i} \leq f_{n,j}$; that is, fitness ranks must be conserved under the transformation, which seems plausible. Furthermore, we distinguish between *analytical-* and *algorithmic* fitness scaling functions. In the former case, ξ_n is an analytical transfer function - e.g. some linear function, parameterized by a chosen slope and offset. Otherwise, ξ_n is an algorithmic procedure without a known analytical definition, e.g. a complex deterministic procedure (say a procedure with many nested conditions in its definition) or a probabilistic procedure. Incidentally, contrary to a deterministic ξ_n, a probabilistic ξ_n generates a distributed $\mathbf{f}_n$ - defined by a vector of means and a vector of variances (and, possibly, vectors of higher statistical moments).

6.1.2 Linear scaling

Linear fitness scaling [29] defines a linear relationship between $f_{n,i}$ and $f_{n-1,i}$ in such a way that the average fitness remains invariant, say $\bar{f}$, under the transformation, while the ratio between the maximal scaled fitness and the average fitness is dynamically kept constant, say s, during the course of a run:

$$\frac{f_{n,max}}{\bar{f}} = s$$

This target ratio, called sensitivity, is a control parameter. The stabilizing effect of linear scaling is that at the early stages of search, when $f_{n-1,max} \gg \bar{f}$, differences in fitness are attenuated in order to prevent premature convergence. On the other hand, at the late stages of search, when $f_{n-1,max} \approx \bar{f}$, differences in fitness are amplified in order to encourage competition between near-optimal strings. At any stage, the competition between strings is proportional to s. The transfer function ξ_n for linear fitness scaling can be defined as:

$$f_{n,i} = \xi_n(f_{n-1,i}) = \frac{\bar{f}}{f_{n-1,max} - \bar{f}} \cdot ($$
$$(f_{n-1,max} - \bar{f}s) + (s-1)\, f_{n-1,i})$$

From this equation, one can readily verify the abovementioned requirements: $f_{n,i} = \bar{f}$ for $f_{n-1,i} = \bar{f}$, and $f_{n,max} = s\bar{f}$ for $f_{n-1,i} = f_{n-1,max}$.

6.1.3 Sigmoid scaling

Sigmoid fitness scaling is gleaned from artificial neural network science [69, 65, 67]. We adopt a sigmoid transfer function, ξ_n, with its inflection point about the average unscaled fitness, $\bar{f}_{n-1}$:

$$f_{n,i} = \xi_n(f_{n-1,i}) = 1 + \tanh\left((s-1)\left(\frac{f_{n-1,i}}{\bar{f}_{n-1}} - 1\right)\right)$$

Basically, sigmoid fitness scaling segregates the population in a worst performing part and a best performing part, while a mild competition among the members within the latter is still meaningfully retained; like in linear fitness scaling, the transformation imposes a lowbound and a highbound on the scaled fitness. The control parameter s ($s > 1.0$) is called the segregation degree. For increasing s values, the sigmoid function approximates a step function. This leads to more stability when e.g. fitness-proportional reproduction is applied, because: (1) the competition between good- and bad strings asymptotically approaches a constant level, so that progress can be made at any stage of the search; (2) the competition between the best performing strings vanishes asymptotically (yet, any remaining trace of competition left is still meaningful).

6.1.4 Power law scaling

Power law scaling [28] exponentially "blows up" (magnifies) differences in fitness; the transfer function, ξ_n, can be defined as:

$$f_{n,i} = \xi_n(f_{n-1,i}) = f_{n-1,i}^{\alpha}$$

where the exponent α is a control parameter.

6.1.5 Sharing: accounting crowding pressure

In essence, sharing [34, 29, 19] attenuates the fitness of an arbitrary string in proportion to the crowding pressure it "feels", i.e. in proportion to the average proximity to other strings in the population. The idea behind this is motivated by nature: overcrowding leads to a situation wherein individuals must either share limited resources or seek other resources elsewhere to have a better chance of survival.

As sharing encourages the dispersal of too similar individuals in the search space, stable subpopulations can emerge on different optima. Sometimes, indeed, the practitioner wants to find more than one adequate solution in a single run, depending on the target problem and/or on his/her objectives. (Without sharing, the formation of stable subpopulations is frustrated by a phenomenon called "genetic drift" [17, 35] – an instability caused by sampling errors related to the finite population size.) The dispersion caused by sharing also opposes premature convergence, which is a pleasant side effect. On the other hand, however, sharing-based fitness scaling, as it features a quadratic time-complexity in the population size (as can be inferred from the definition below), is computationally quite expensive.

The fitness scaling function ξ_n for sharing can be defined as:

$$f_{n,i} = \xi_n(f_{n-1,i}) = \frac{f_{n-1,i}}{\sum_{j=1}^{N} \left\{ \begin{array}{ll} 1 - \left(\frac{d_{i,j}}{\delta}\right)^{\alpha} & , \text{if } d_{i,j} < \delta \\ 0 & , \text{otherwise} \end{array} \right\}}$$

where N is the population size, δ is a critical distance that controls the spatial extent of sharing, and α is an exponent that regulates the sensitivity to the string-string distances $d_{i,j}$. The expression in the denominator between braces, $\{\cdots\}$, is a power law function; it has been chosen for its provable adequacy in simulating limited resources. In passing, note that ξ_n is algorithmic, as its definition contains conditions.

Sharing-based fitness scaling distinguishes itself from many other fitness scaling strategies in that it interprets strings, as it must do in order to calculate string-string distances according to some predefined criterion. When the distances are calculated in the encoded space – e.g. based on a Hamming metric – one speaks of genotypic sharing; when distances are calculated in the decoded space – e.g. based on an Euclidean metric – one speaks of phenotypic sharing. The computational overhead of sharing may be reduced by estimating $\sum_{j=1}^{N}\{\cdots\}$ from a sample of strings in the population; a smaller sample results in a larger reduction in the overhead, but also in a larger scaling error.

6.1.6 Normalization

Normalization does not affect the competition between strings, but is nonetheless required for certain reproduction- and replacement procedures, as pointed out above. The normalization function, ξ_n, is defined as:

$$f_{n,i} = \xi_n(f_{n-1,i}) = \frac{f_{n-1,i}}{\sum_{i=1}^{N} f_{n-1,i}}$$

where N is the population size.

6.1.7 Example

A typical example of fitness scaling performed in a sequence of elementary fitness scaling steps is: (1) translation; (2) sigmoid scaling; (3) normalization.

6.2 String reproduction

It is convenient to view string reproduction as a special fitness scaling step – one that is appended to the preceding fitness scaling steps. It is special in that it is always algorithmic – i.e. the scaling algorithm used is a string selection criterion – and always implicit; by the latter we mean that the scaled fitnesses are not actually calculated and stored as a vector in memory, but, instead, manifest themselves actively as selection rates – or, equivalently, as reproduction rates – that can be observed. Consistency demands that the selection rate of a particular string may not be lower than the selection rate of another string that has a lower (scaled) fitness, and *vice versa*, otherwise the search is encouraged in the wrong direction.

One may distinguish "full reproduction" and "partial reproduction". In order to understand this distinction, recall that any reproduction procedure is applied according to the following general scheme: a selection criterion is applied N' times to a population of N strings, whereby N' is a control parameter that satisfies $2 \leq N' \leq N$; each selected string is copied to a temporary population, which therefore has size N' after reproduction is completed. (The minimal value of N' is 2, because recombination can not be applied for a smaller N'.) Full reproduction refers to the case wherein $N' = N$, whereas partial reproduction refers to the case wherein $N' < N$. Genetic algorithms with full reproduction (hence full replacement) have no overlapping populations, i.e. no generation gap, and are called generational genetic algorithms. Genetic algorithms with partial reproduction (hence partial replacement) have overlapping populations, i.e. generation gap, and are called steady state genetic algorithms; genetic algorithms of this kind enjoy increasing interest [88, 80, 87, 22, 20, 15, 82, 30].

Selection criteria can be combined into a so-called *sequential compound selection criterion.* Such a criterion uses one selection criterion for the first N'_1 string selections during reproduction, another selection criterion for the next N'_2 string selections, and so on if there are more constituent selection criteria, whereby $N'_1, N'_2, \cdots$ are control parameters that satisfy $N'_1 + N'_2 + \cdots = N'$ – the size of the temporary population.

Selection criteria can also be combined into a so-called *nested compound selection criterion.* Such a criterion selects a string from a set of strings (rather than from the population) selected earlier by iterating another selection criterion a number of times. Deeper nesting is possible, but computational demands increase exponentially with the number of nesting levels.

A simple selection criterion is understood to process either fitness values or fitness ranks, but not both. (The practical advantages of rank-based selection criteria are outlined in Section 4.4.2 above.) Compound selection criteria may process both fitness values and fitness ranks. Some examples are given in the next subsections.

6.2.1 Random selection

The random selection criterion arbitrarily selects a string, irrespective of known fitness values. Reproduction based on random selection seems somewhat awkward from a traditional point of view (since no selection pressure is exerted), but it is not objectionable provided that selection pressure is exerted in the replacement stage, as pointed out above (see Section 4.8.1).

6.2.2 Roulette selection

The roulette selection criterion selects a string with a probability proportional to its scaled fitness value; technical details are described in [55]. Among other reproduction schemes, fitness-proportional, or simple, reproduction (Part 1 [56]) has the longest record of application.

6.2.3 Linear selection

The linear selection criterion [1, 2, 3, 38, 87] selects a string with a probability proportional to its fitness rank rather than -value. That is, with ranks ranging from, say, n (worst) to $n + N - 1$ (best), the string with rank r is selected with probability $\frac{r}{Nn+N(N-1)/2}$, where N is the population size and n is an offset (non-negative integer control parameter). For $n = 0$ the worst string is never selected and selection pressure is maximal.

6.2.4 Exponential selection

The exponential selection criterion [80, 82] selects the best string with probability p; if this fails, it selects the second best string with probability p; if this fails, it selects the third best string with probability p; and so on; p is a control parameter. (If none of the strings are selected, the procedure is simply repeated.) In this way, with ranks ranging from, say, n (worst) to $n+N-1$ (best), the string with rank r is selected with probability p^{r+1-n}.

6.2.5 Elitist selection

The elitist selection criterion [29] selects the best string the first time it is called; it selects the second best string the second time it is called; it selects the third best string the third time it is called; and so on. This deterministic selection criterion is parameterized by N_1' - the number of calls after which it will be reset so that the above procedure can be repeated for another reproductive session. It is normally applied in a sequential compound selection criterion, with the intent to ensure that the best strings are reproduced into the population of the next generation.

6.2.6 Threshold selection

The threshold selection criterion [55] randomly selects a string among the N_{best} best performing strings, where N_{best} is an integer control parameter ($2 \leq N_{best} \leq N$). That is, strings with a fitness rank below the threshold, $N - N_{best}$, stand no chance of being selected. Selection pressure is zero for $N_{best} = N$ (random selection, basically) and maximal for $N_{best} = 2$. (Note that $N_{best} = 1$ would cause zero diversity in the population, which is undesirable; it is therefore excluded from the definition.)

The threshold selection criterion may be viewed as a nested compound selection criterion: the random selection criterion applied to a set of strings selected earlier by iterating the elitist selection criterion $N_1' = N_{best}$ times.

6.2.7 Tournament selection

The tournament selection criterion [9, 30] selects the best string from M randomly selected strings, where M is an integer control parameter ($1 \leq M \leq N - 1$). Selection pressure is zero for $M = 1$ and maximal for $M = N - 1$. (Note that $M = N$ would cause zero diversity in the population, which is undesirable; it is therefore excluded from the definition.) Binary tournament selection ($M = 2$) is widely applied.

The tournament selection criterion may be viewed as a nested compound selection criterion: the elitist selection criterion for $N_1' = 1$ applied to a set of strings selected by iterating the random selection criterion M times. (Note that from the viewpoint of nested compound selection, tournament selection loosely resembles the opposite of threshold selection.)

6.3 String pairing

In preparation for recombination, the N' strings in the temporary population are paired; if N' is odd, then the last string gets no partner and will simply pass recombination unmodified. The assignment of the string pairs may be viewed as the determination of a permutation with length N', using some criterion - the pairing criterion. In this so-called *pairing permutation*, then, the first and second element indicate the first string pair, the third and fourth string index indicate the second string pair, and so on.

6.3.1 Random pairing

Random string pairing is defined by the creation of a random pairing permutation. Although widely applied, this strategy is naive in that it does not include measures against the pairing of strings which will result in unproductive recombination.

6.3.2 Intelligent pairing

Intelligent pairing criteria include constraints such as to encourage the pairing of strings which will

result in productive recombination. These constraints are normally based on string dissimilarities; for, more dissimilar parent strings are more likely to result in productive recombination.

An example of an intelligent pairing strategy is one that encourages the pairing of strings with different fitness values. This seems sensible, since strings with different fitness values must be dissimilar. (Note that the opposite is not necessarily true: dissimilar strings may have the same fitness.) On the other hand, however, the approach is somewhat sloppy in that a difference between fitness values does not uniquely determine a dissimilarity between strings; the opposite is also true: a dissimilarity between strings does not uniquely determine a difference between fitness values.

Another intelligent pairing criterion – one that avoids making decisions on the basis of difference between fitness values, but rather looks directly at dissimilarities between strings to that end – is, for instance, one that prevents "incest", as Eshelman puts it [22, 20]. This strategy first randomly pairs the strings in a fully reproduced population ($N' = N$). Next, it discards the pairs in which the strings are separated by a distance less then some threshold ($N' \leq N$, N' is dynamical); any reasonable distance measure can be used, e.g. Hamming distance (in the encoded space) or Euclidean distance (in the decoded space). The threshold is initially set to the average distance between strings in the initial population, and then decremented dynamically as the population converges and thus looses diversity; a decrement is made each time that there would otherwise be no more non-incestuous pairs of parent strings left for recombination.

In both intelligent pairing strategies described above, productive recombination is merely encouraged, but not guaranteed.

6.4 String replacement

String replacement proceeds along lines reminiscent of string reproduction: a string selection criterion is applied N' times to a population of N strings, whereby $2 \leq N' \leq N$. Each selected string is replaced by a string withdrawn from the temporary population, which initially has size N'. (Selection criteria for string withdrawal from the temporary population are beyond the purposes of this tutorial part; random withdrawal may be assumed for simplicity.)

One may distinguish "full replacement" ($N' = N$) and "partial replacement" ($N' < N$); in the former case, there is obviously only one way in which the replacement can be accomplished; in the latter case, several possibilities exist, depending on the selection criterion used. Thus, partial replacement can be a means to control exploitation, as pointed out above (see Section 4.8.1).

Importantly, any selection criterion that can be used for string reproduction (see Section 6.2 above) can, in principle, also be used for string replacement, provided that some inverting transformation is applied to the fitness values; as such a transformation reverses fitness ranks, one sometimes speaks of reversed fitness values [82]. Consistency demands that the selection rate of a particular string may not be lower than the selection rate of another string that has a higher (scaled) fitness, and *vice versa*, otherwise the search is encouraged in the wrong direction.

An important consequence of partial replacement/reproduction is that parent strings and child strings may coexist, because parents are not necessarily replaced after a generation. This provides the opportunity that a particular string receives more trials to act as a parent than would be possible in generational genetic algorithms (genetic algorithms that employ complete replacement/reproduction); such can be strongly encouraged by means of elitist selection applied in either the reproduction procedure or replacement procedure, or both. Moreover, the child strings themselves are immediately available to become parents themselves, i.e. there is no need for them to wait to become parents until N strings have been reproduced. The supposed advantage of this has been phrased by Syswerda as follows [82]: "By the time generational GAs start using good material, steady-state genetic algorithms have already thoroughly integrated the material into the population."

7 Configuration strategies

Starting from Section 4.5, in this section we outline a number of practicable strategies for genetic configuration.

7.1 Control parameterization

This section focuses on control parameterization: the setting of numerical control parameters. The lines of approach discussed in the following subsections are most commonly used in practice [15].

7.1.1 Skilful hand optimization

In this approach the practitioner first applies common wisdom – e.g. rules of thumb available in the literature – to make a rough estimate of the optimal control setting. Next, this estimate is refined through step-by-step, manual adjustments guided by insight from experience.

Some recommended rules of thumb when using bit-level exploration operators are: (1) take the probability for mutation (p_m) below 0.05; (2) take the probability for recombination (p_r) between 0.5 and 1.0; (3) for **B_UX**, take the target bit swap rate (p_s) between 0.2 and 0.4. Likewise, some rules of thumb are available for control parameters that concern exploitation.

It follows from practice that skilful hand optimization amounts to the fastest and most reliable way to achieve good configurations, or, as Michalewicz puts it [61]: "It seems that finding good values for the GA parameters is still more an art than a science". Indeed, the shortcomings of automated (criterion-based) approaches, discussed in the next sections, are generally deemed serious. On the other hand, admittedly, an important shortcoming of skilful hand-optimization is that the practitioner can often not rationally explain the choices made.

7.1.2 Systematic search

An example of an automated approach of this kind is a full factorial design, or grid search, in the search space defined by the set of control parameters of interest. The obvious advantage of this strategy is that it guarantees that the optimal control setting will ultimately be found. The disadvantage, however, is that the computational burden grows exponentially with the number of control parameters. In many cases this implies that the task can not be accomplished within practically reasonable time. For an impression of the required running time: using a relatively small set of control parameters, Schaffer *et al.* [70] conducted a full factorial design which required more than one CPU year of a fast computer.

In fractional factorial designs [4, 25, 41], a drastic reduction in the number of required experiments is obtained by making assumptions about the parameters, e.g. that they are not correlated. Such assumptions are not legitimate for genetic control parameters, as these are strongly correlated, in general (see Section 4.5 above). The benefit of fractional factorial designs should therefore be sought in the fact that they provide a rough estimate of the optimal control setting to the practitioner which he/she can subsequently optimize by hand.

7.1.3 Meta-optimization

In this automated approach, the control parameters of a genetic algorithm are optimized by another optimization method – a meta-optimizer. Grefenstette used a genetic algorithm to meta-optimize the control setting of a simple genetic algorithm [36]. He adopted a test suite of objective functions due to De Jong [17] and obtained control settings that (slightly) outperformed those that De Jong had found by way of hand optimization [17]. Other investigations involving meta-optimization by a genetic algorithm were conducted by Bramlette for non-binary representations [8].

The use of a robust search technique for meta-optimization (e.g. another genetic algorithm) is likely to result in an acceptable control setting and to be computationally less intensive than grid search. Nonetheless, given practically reasonable time constraints, the procedure can only be carried out when a limited number of control parameters are taken into account, as a rule; the practitioner will choose the most important control parameters to his/her best judgment, and keep the others at a constant value during the parameterization procedure.

7.1.4 Dynamic parameterization

The preceding approaches to control parameterization aim at finding an optimal setting that does not change during a run. However, it is reasonable to assume that if genetic configuration is to stay optimal during genetic search, then, ideally, it should evolve dynamically in response to the

evolving population. This demands dynamic control setting on a per-generation basis. It can be achieved either by way of a preprogrammed adjustment schedule or adaptively.

A preprogrammed adjustment schedule for a particular set of control parameters is imposed in a fashion similar to a "cooling" schedule in simulated annealing (Part 1 [56]). A serious drawback of this approach, however, is that it merely replaces the search space of candidate control settings by a (possibly larger and/or more complex) search space of candidate adjustment schedules among which an adequate schedule must be found.

Adaptive dynamic control setting was most noticeably pioneered by Davis [13] and seems to be more flexible. There is an on-line monitoring criterion which, on the fly, observes how well each heuristic operator has been doing recently. Intermittently, an adjustment mechanism updates a set of weights associated with the control parameters in response to the observations. An advantage of the on-line adaptation strategy over the aforementioned control parameterization strategies, is that it consumes only a comparatively small amount of computational resources. However, the difficulty lies in selecting a good performance criterion.

Li *et al.* [46] report a dynamic genetic algorithm (for wavelength selection in NIR data) that uses an ANOVA-based monitoring criterion for on-line adjustment of control parameters.

In view of the strong correlations between the control parameters, dynamic parameterization can lead to reliable results only when a limited number of control parameters is taken into account, as a rule. The practitioner will choose the most important control parameters to his/her best judgment, and keep the others at a constant value during a run; examples of control parameters that are likely to be kept constant are the center values, c_i, and deviations, d_i, that define the search volume (see Section 5.2.1 above).

7.2 Empirical complexity analysis

A promising approach towards optimizing genetic configuration borrows from a methodology recently introduced: the empirical analysis of problem complexity, based on an explorative walk in the search space (Part 1 [56], [55]). The time-series of observations performed during the walk is subjected to a (auto)correlation analysis in order to obtain a statistical estimate that quantifies the ruggedness of the o-landscape. When the search heuristics and their parameterization are chosen in such a way that this estimate – the apparent problem complexity – becomes minimal, optimal configuration has been achieved.

Manderick *et al.* [58] investigated the separate optimization of the recombination module and the mutation module, using independent empirical complexity analyses. In this way, partial genetic configurations are obtained which turn out to be acceptable (i.e. near-optimal) components in an overall genetic configuration. Hence, once exploration modules have been optimized separately and are kept fixed, the exploitation modules can be optimized simultaneously next. This sequential strategy is promising, since the high-dimensional configuration problem can be subdivided into a number of low-dimensional (i.e. partial) configuration problems – leading to an exponential reduction in the overall computational burden of genetic configuration.

Incidentally, in a cursory study that we have performed, involving an artificial problem (*viz.* fitting a Fourier function) in which the problem parameters are strongly correlated, we observed a significant reduction in the apparent problem complexity when switching from **B MX** to **B UX**; this is consistent with expectations pointed out in Section 5.2.2 above that **B UX** is normally a better choice when problem parameters are strongly correlated.

8 Hybridization strategies

As was noted in Part 1 [56], the performance of a genetic algorithm may be enhanced by hybridizing (combining) its search heuristics with the search heuristics of another method – either sequentially or in parallel. Hybridization enables one to achieve a reduction in the overall computational load and to obtain more accurate and/or -precise end-solutions. Objectives of this kind are mainly economically motivated, i.e. one may save time and/or money when they are accomplished.

Hybridization strategies can be categorized as follows:

- Sequential hybridization:
  - Pre-hybridization

- Post-hybridization
- Self-hybridization (reconfiguration)

- Parallel hybridization:
  - Problem partitioning
  - Population partitioning
  - Incorporation of domain-specific search heuristics

8.1 Pre-hybridization

Sometimes an inaccurate yet not entirely useless estimate of the solution of the problem under consideration can be obtained from some knowledge source – either human or system. We consider two ways in which a genetic algorithm can utilize such prior knowledge.

In the first approach, the initial population is constructed in such a way that it contains a few copies of a string that represents the initial estimate. The advantage of such a "seeding" strategy is that it is universal (domain-independent), i.e. it can be implemented in any genetic algorithm, irrespective of the target problem.

In the second approach (Figure 11), which is restricted to NUM problems, the initial estimate represents the working point: the center of a search volume of which the deviations are chosen by the user.

In choosing the dimensions of the search volume the practitioner faces the dilemma that a smaller search volume results in a potentially shorter convergence time, but also in a higher risk to accidentally exclude the true solution from the search volume.

De Weijer *et al.* [86, 57] report a genetic algorithm for curve-fitting that searches around a working point determined by an artificial neural network.

Besides pre-hybridization, genetic algorithms may incorporate problem-specific knowledge in other ways as well, e.g. in exploration operators (see Section 4.6 above); for more general treatises on this subject, the reader is referred to [37, 29, 15, 61].

8.2 Post-hybridization

Post-hybridization is concerned with the improvement, or refinement, of the end-solution found by a genetic algorithm (Figure 11). This serves to remedy an important shortcoming of genetic algorithms: their comparatively poor search precision (Part 1 [56]). A local search technique is used for the improvement; thus, the genetic algorithm used may be regarded as a hill-finder and the local search technique subsequently invoked, as a hill-climber.

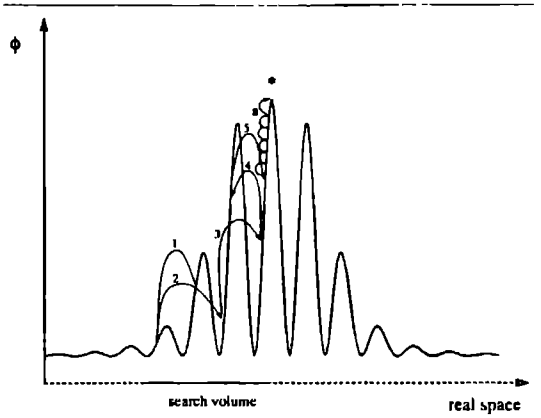

Figure 11: The choice of the search volume in the real space is an example of pre-hybridization; prior knowledge that has been used here is that the fitness landscape is symmetrical. The search volume is successful as it contains the global optimum (*). A genetic algorithm that performs search (1,2,3,4,5) to obtain a rough solution, is post-hybridized by a search method that refines this solution through local search (a) in order to find the global optimum.

Blommers *et al.* [6] report a genetic algorithm for conformational analysis followed by refinement through energy minimization based on steepest descent and conjugate gradient search.

In case that the genetic algorithm is combinatorial, its result can be refined by a combinatorial local search method. In many combinatorial local search methods, a step in the search space is performed by first calculating for each pair of k-nearest neighboring elements (in the string that represents the candidate solution) the gain in performance if elements would swap, and then accepting the trial which has yielded the highest gain; usually $k = 1$, but other values or even value ranges may be useful.

8.3 Self-hybridization (reconfiguration)

When a genetic algorithm is pre- or post-hybridized with another, differently configured genetic algorithm, the approach is called self-hybridization, or reconfiguration.

A sensible extension of this idea is iterated reconfiguration. This results in a chain of different genetic algorithms (GAs), each of which – except the last – passes a result to its successor for further improvement:

$$\text{domain data} \rightarrow \text{GA1} \rightarrow \text{GA2} \rightarrow \text{GA3} \rightarrow \cdots \cdots \rightarrow \text{end-result}$$

The next subsections outline four approaches to iterated reconfiguration for genetic algorithms applied to NUM problems.

8.3.1 Iterated re-parameterization

In iterated re-parameterization, each successive genetic algorithm in the chain uses a new set of values for its control parameters. The user is free to choose which control parameters are updated each iteration, and by what new values. In this way, he/she can exert substantial influence in a highly interactive fashion during the entire procedure; if practised skilfully, this can be an important advantage over dynamic parameterization (see Section 7.1.4 above).

Special cases of iterated re-parameterization are obtained when only certain control parameters are subjected to the updating. For instance, in iterated element expansion (Section 8.3.2 below), element sizes are iteratively updated; in iterated search grid expansion (Section 8.3.3 below), bit-field sizes are iteratively updated; and in iterated search volume contraction (Section 8.3.4 below), deviations and center values that define the search volume are iteratively updated.

8.3.2 Iterated element expansion

In iterated element expansion, each successive genetic algorithm in the chain uses exploration operators that manipulate elements that have become larger. (These elements can be viewed as virtual digits, see Figure 4.) This is done in such a way that the string format is not affected; hence, less elements per string are manipulated since the string size remains constant. Iterated element expansion is consistent with the fact that less exploration is required in later stages of the search. From the viewpoint of information processing, the imposed stepwise growth of elements swapped upon recombination, is consistent with the gradual growth of Holland building blocks and thus may increase the overall search efficiency.

8.3.3 Iterated search grid expansion

Iterated search grid expansion, or *dynamic parameter encoding* due to Schraudolph and Belew [71], comes down to a step-by-step augmentation of the encoding resolutions (ρ_i) for some problem parameters (x_i) so that each successive genetic algorithm in the chain searches on a larger grid, i.e. a larger search space (Figure 12). Given that the search

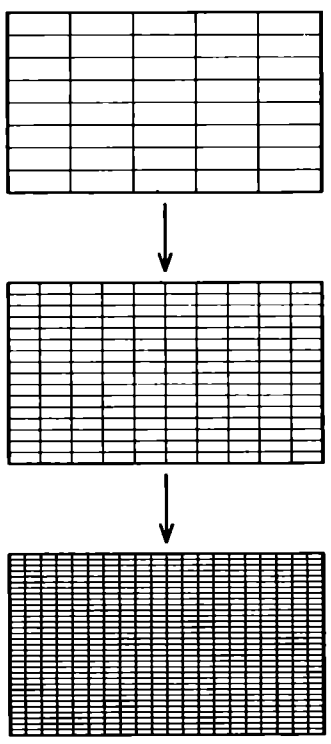

Figure 12: Iterated search grid expansion.

volume is thereby not affected, the respective decoding resolutions ($\rho_i/(2d_i)$, d_i constant) are increased. In this way, each successive genetic algorithm in the chain gains a higher search precision to refine the result of its predecessor.

The computational rationale for dynamic parameter encoding is the following. Normally, the genetic modification heuristics (exploration heuristics) pay equal attention to all bits in the strings. Evidently, this can not be an optimal strategy because values for the least significant bits in the bitfields in a string only make sense when the values of the most significant bits are already determined. Dynamic parameter encoding avoids wasting computational effort in evaluating the least significant bits by simply excluding them explicitly from participation in the early stages of the search. The procedure is exemplified as follows.

For problem parameter x_i, suppose the bitfield is 101. The length of this bitfield is $\ell_i = 3$, its decimal value is $D_i = 5$, and the resolution for x_i is $\rho_i = 2^{\ell_i}$. A simple way to increase ρ_i is to double it: $\ell_i = 4$; that is, to add a bit to the bitfield. (Note that thereby the string format is changed, i.e. the length of all population strings is incremented by one bit.) The bit is added as a 0-bit at the righthand side of the bitfield: 1010. This "right-append-a-0" operation is called bitfield recoding. It doubles not only ρ_i, but D_i as well: $D_i = 10$. Accordingly (by Equation 3), the decoded real value of x_i is unaffected, which is convenient. Summarizing, the population of each successive genetic algorithm in the chain is initiated by recoding the population of its predecessor.

8.3.4 Iterated search volume contraction

Iterated search volume contraction, or *delta coding* due to Whitley *et al.* [89, 60], comes down to a step-by-step decrease of the deviations (d_i) for some problem parameters (x_i), i.e. to a step-by-step pruning of the search volume (Figure 13). Given that the search grid is thereby not affected, the respective decoding resolutions ($\rho_i/(2d_i)$, ρ_i constant) are increased. In this way, again, each successive genetic algorithm in the chain gains a higher search precision to refine the result of its predecessor.

The working point for each genetic algorithm in the chain is a provisionally best estimate of the solution. For the first genetic algorithm, it is provided by the user, e.g. a random guess. The working point for each successive genetic algorithm in the chain is provided by its predecessor.

Note that iterated search volume contraction relies considerably on the widely praised tendency of genetic algorithms to approach the global optimum: indeed, owing to this property (good search accuracy) the search volume can be pruned without too much risk to accidentally exclude the global optimum.

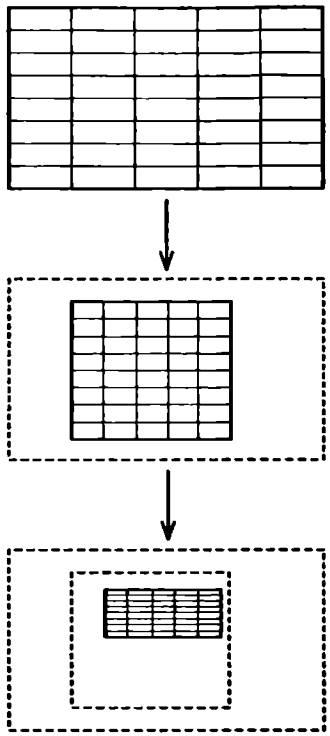

Figure 13: Iterated search volume contraction.

The population of each genetic algorithm in the chain is usually initiated randomly. An important advantage of this is that diversity is introduced after each pruning step (Figure 14), which avoids premature convergence and in many cases obviates mutation as well.

De Weijer *et al.* [86, 57] report an application of iterated search volume contraction for curve-fitting.

8.4 Problem partitioning

Problem partitioning makes sense when the problem of interest basically comprises a number of smaller, independent problems. The set of problem parameters is partitioned accordingly (e.g. the torsion angles for supposedly non-interacting func-

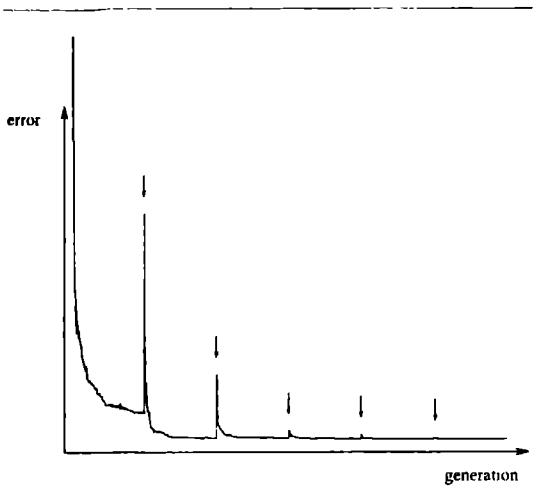

Figure 14: Typical performance plot obtained after iterated search volume contraction. ↓s indicate restarts of the genetic algorithm with a randomly initiated population. Both the search accuracy and search precision gradually improve.

tional groups in a large molecule) such that independent genetic searches can be conducted for each separate partition. The partial solutions of these searches are subsequently combined to obtain an estimated total solution. The gain of such a strategy can be tremendous, as the total running time decreases exponentially with the number of problem partitions chosen.

In real-world problems, however, entirely independent problem partitions usually do not exist. Hence, the estimated total solution may not be accurate. In this case, subsequent improvement of the solution by a genetic algorithm, or another search technique for that matter, can be rewarding.

8.5 Population partitioning

When the population is divided into smaller subpopulations and at least one operator in the evolution cycle (Figure 1) is executed independently in each of these, one obtains a partitioned genetic algorithm. A practically attractive example of a partitioned genetic algorithm is one in which string evaluation is performed in parallel using separate processors, each of which accommodates a different subpopulation [79]; these processors are called slaves, as they are controlled by a master processor that runs the genetic algorithm cycle. Such a scheme makes sense, since for many real-world problems, string evaluation is computationally most costly, as a rule. In the extreme case, a processor is allocated to each string separately (provided that enough processors are available). The parallel approach is possible and can be implemented comparatively easily, because any string can be evaluated independently from the others.

Another approach to partitioning a genetic algorithm is obtained when all operators in the evolution cycle are executed independently in each of the subpopulations, except for a new operator – migration. Migration intermittently selects promising strings from each subpopulation and migrates them to a different subpopulation [83, 78, 90]. This special case of a partitioned genetic algorithm is called a distributed genetic algorithm. The rationale for a distributed genetic algorithm is that by occasionally swapping strings in this way, new high-quality genetic material can be introduced in the evolving subpopulations, thus improving the sampling properties of the entire searching system.

In a sense, migration resembles recombination: while recombination swaps elements between strings, migration swaps strings between subpopulations; accordingly, migration has been called "meta-recombination" on some occasions. The operator is controlled by two parameters: the migration interval (the number of generations between each migration) and the migration rate (the proportion of strings selected from each subpopulation for migration at the time of migration).

8.6 Incorporation of domain-specific heuristics

The idea to integrate established, domain-specific (greedy) heuristics into genetic algorithms to take care of the exploration, has crossed the mind of many practitioners who in daily practice are concerned with one particular problem only. These individuals are usually prepared to sacrifice any excess domain-independency if that leads to considerable savings in the time it takes to find acceptable solutions, as normally is the case (see Section 4.6). Thus, problem-specific exploration operators make most sense in commercial (rather than scientific) settings, where savings in time

usually imply savings in money.

Elaborate treatises on the incorporation of domain-specific heuristics into genetic algorithms are found in [15, 61]; an example of chemometric interest is described by Kelly and Davis [42].

9 Conclusions

We conclude both parts of this tutorial with a summary of the salient properties of genetic methodology.

Intuitive appeal. In practice, the *intuitive appeal* of a method often plays a key role in its dissemination among potential users. This requires a simple algorithmic structure based on simple principles. The intuitive appeal of genetic algorithms is widely acknowledged, in general; they are based on conceptually simple, generally applying rules for the improvement of performance: a competition rule ("struggle for life") and a selection rule ("survival of the fittest"); furthermore, they are structurally simple. These attractive properties render genetic algorithms comparatively easy to implement, too.

Mechanical complexity. Although the separate actions that take place in a genetic algorithm are comparatively simple on their own, the overall evolution process is mechanically complex due to the large amount of procedural interactions that take place through the population – the central medium of all procedural actions. In general, it is this *mechanical complexity* that makes it so difficult to configure a genetic algorithm in such a way that two important conditions for good performance are fulfilled: (1) the exploration proceeds in accordance with the building block principle, and (2) the rates of exploration and exploitation are balanced.

Poor accessibility. The configuration of a genetic algorithm amounts to an instantiation of the aforementioned generic flowchart, or: the selection, implementation and parameterization of the constituent procedures, or routines. Optimizing configuration amounts to a laborious task due to the mechanical complexity in combination with the immense size of the configuration space; for a more complex target problem, the situation normally becomes even harder to deal with. It may be ascribed to this coincidence, i.e. *poor accessibility*, that reliable and efficient standard methodology for optimal configuration is not available. Actually, the best configuration strategies appear to be those based on a good understanding of- and experience with genetic algorithms; some practical rules of thumb – mostly involving the control parameterization side of configuration – are available in the literature and provide some guidance. Summarizing, genetic methodology is especially poorly accessible to novice practitioners.

Versatility. Innumerable recombination routines, mutation routines, selection routines, fitness scaling routines, etc. are conceivable; numerous among these have found (and many more probably shall find) frequent implementation. Together with objective functions for particular problems, all possible combinations amount to innumerable instantiations of the generic flowchart. This configurational flexibility, or *versatility*, reflects the ability of genetic methodology to adapt to very divergent problems (depending on the practitioner's choices). On the other hand, however, the versatility often strikes the novice practitioner as overwhelming.

Power. The power of genetic algorithms resides in their efficiency and robustness. The efficiency of genetic algorithms is due to exponential search behavior, as explicited in the schema theorem [39, 29, 56] – the mathematical foundation of genetic algorithms.

Genetic algorithms are most robust when domain-independent exploration operators are used. The robustness manifests itself in several senses, i.e. in that the search is: (1) not easily misled by sub-optima on the fitness landscape; (2) not easily misled by fitness inconsistencies that are a result of small artefacts in the domain (e.g. missing input data or noise in input data); (3) not dramatically affected by small changes in configuration; related to that: (4) comparatively indifferent to different population initiations.

Search accuracy and -imprecision. Owing to their robustness, genetic algorithms are characterized by a good search accuracy: the tendency to approach the global solution under all among

many different circumstances. On the other hand, however, genetic algorithms appear to exhibit a poor search precision: a considerable spread in estimated solutions about the global optimum upon replicating runs; this spread is to be ascribed to the "method noise" generated by the different probabilistic components.

Hybridizability. Genetic algorithms lend themselves excellently for further enhancement of their power through hybridization with other problem solving strategies. An example of hybridization is the use of local search algorithms – either on-line as domain-dependent exploration routines, or off-line such as to either provide an initial estimate (pre-hybridization) or improve an ultimate estimate (post-hybridization).

Parallelizability. Given that each string in the population can be evaluated independently from the other strings, genetic algorithms are highly suitable for explicit parallelization: computation is distributed across a host processor and several slave processors; each slave processor evaluates the strings in a small sub-population – possibly as small as one string. Importantly, the parallelization can be accomplished without specialized parallel hardware such as transputer systems that make memory access and communication fast. The reason for this is that string evaluation is usually slower, i.e. determines the overall speed of execution. An example of a parallel system of processors that would suffice is a local area network of workstations; the software required for the communication is comparatively simple.

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007. We are endebted to A.D. Dane for his aid in the design of the mix subset recombination operator.

References

[1] Baker, J.E. Adaptive selection methods for genetic algorithms. In Grefenstette, J.J., editor, *Proceedings of the First International Conference on Genetic Algorithms*, page 101, Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[2] Baker, J.E. Reducing bias and inefficiency in the selection algorithm. In Grefenstette, J.J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 14, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[3] Baker, J.E. *Analysis of the Effects of Selection in Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, TN, 1989.

[4] Bayne, C.K. and Rubin, I.B. *Practical Experimental Designs and Optimization Methods for Chemists*. VCH Publishers, Deerfield Beach, FL, 1986.

[5] Bhuyan, J.N., Raghavan, V.V., and Elayavalli, V.K. Genetic algorithms for clustering with an ordered representation. In Belew, R.K. and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 408–415, San Mateo, CA, 1991. Morgan Kaufmann.

[6] Blommers, M.J.J., Lucasius, C.B., Kateman, G., and Kaptein, R. Conformational analysis of a dinucleotide photodimer with the aid of the genetic algorithm. *Biopolymers*, 32:45–52, 1992.

[7] Booker, L.B. Improving search in genetic algorithms. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, page 61. Morgan Kaufmann, Los Altos, CA, 1987.

[8] Bramlette, M.F. Initialization, mutation and selection methods in genetic algorithms. In Belew, R.K. and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 100, San Mateo, CA, 1991. Morgan Kaufmann.

[9] Brindle, A. *Genetic Algorithms for Function Optimization*. PhD thesis, University of Alberta, Edmonton, AB, 1981.

[10] Cartwright, H.M. and Mott, G.F. Looking around: Using clues from the data space to guide genetic algorithm searches. In Belew, R.K. and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 108–114, San Mateo, CA, 1991. Morgan Kaufmann.

[11] Caruana, R.A. and Schaffer, J.D. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In Laird, J.E., editor, *Proceedings of the Fifth International Conference on Machine Learning*, page 153, Los Altos, CA, 1988. Morgan Kaufmann.

[12] Davis, L Applying adaptive algorithms to epistatic domains In *Ninth International Joint Conference on Artificial Intelligence*, pages 162–164, 1985

[13] Davis, L Adapting operator probabilities in genetic algorithms In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69, San Mateo, CA, 1989 Morgan Kaufmann

[14] Davis, L , editor *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York, NY, 1991

[15] Davis, L , editor *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York, NY, 1991 Part I A Genetic Algorithms Tutorial, p 1–101

[16] De Jong, K A Personal communication, First Workshop on Parallel Problem Solving from Nature, University of Dortmund, Germany, October 1 3, 1990

[17] De Jong, K A An analysis of the Behavior of a Class of Genetic Adaptive Systems Technical Report 185, University of Michigan, Ann Arbor, MI, August 1975 Ph D Thesis

[18] De Jong, K A and Spears, W M An analysis of the interacting roles of population size and crossover in genetic algorithms In Schwefel, H -P and Manner, R , editors, *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, page 38, Berlin, 1991 Springer-Verlag Lecture Notes in Computer Science 496

[19] Deb, K and Goldberg, D E An investigation of niche and species formation in genetic function optimization In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 42, San Mateo, CA, 1989 Morgan Kaufmann

[20] Eshelman, L J The CHC adaptive search algorithm How to have save search when engaging in nontraditional genetic recombination In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, page 265, San Mateo, CA, 1991 Morgan Kaufmann

[21] Eshelman, L J , Caruana, R A , and Schaffer, J D Biases in the crossover landscape In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, San Mateo, CA, 1989 Morgan Kaufmann

[22] Eshelman, L J and Schaffer, J D Preventing premature convergence in genetic algorithms by preventing incest In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 115, San Mateo, CA, 1991 Morgan Kaufmann

[23] Falkenauer, E A genetic algorithm for conceptual clustering Technical Report Report FMS39, CRIF Industrial Automation, Brussels, December 1991

[24] Falkenauer, E A genetic algorithm for grouping In Gutierrez, R and Valderrama, M J , editors, *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis*, pages 198–206, Singapore, 1991 World Scientific

[25] Fedorov, V V *Theory of Optimal Experiments*, volume 12 of *Probability and Mathematical Statistics* Academic Press, New York, NY, 1972 Originally published in Russian

[26] Fogarty, T C Varying the probability of mutation in the genetic algorithm In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 104, San Mateo, CA, 1989 Morgan Kaufmann

[27] Fox, B R and McMahon, M B Genetic operators for sequencing problems In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, pages 284–300, San Mateo, CA, 1991 Morgan Kaufmann

[28] Gillies, A M *Machine Learning Procedures for Generating Image Domain Feature Detectors* PhD thesis, University of Michigan, Ann Arbor, MI, 1985

[29] Goldberg, D E *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, Reading, MA, 1989

[30] Goldberg, D E and Deb, K A comparative analysis of selection schemes used in genetic algorithms In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, page 69, San Mateo, CA, 1991 Morgan Kaufmann

[31] Goldberg, D E , Deb, K , and Korb, B Don't worry, be messy In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 24, San Mateo, CA, 1991 Morgan Kaufmann

[32] Goldberg, D E , Korb, B , and Deb, K Messy genetic algorithms Motivation, analysis, and first results *Complex Systems*, 3 493, 1989

[33] Goldberg, D E and Lingle, R Alleles, loci, and the traveling salesman problem In Grefenstette, J J , editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, 1985 Lawrence Erlbaum Associates

[34] Goldberg, D E and Richardson, J Genetic algorithms with sharing for multimodal function optimization In Grefenstette, J J , editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 41, Hillsdale, NJ, 1987 Lawrence Erlbaum Associates

[35] Goldberg, D E and Segrest, P Finite markov chain analysis of genetic algorithms In Grefenstette, J J , editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 1, Hillsdale, NJ, 1987 Lawrence Erlbaum Associates

[36] Grefenstette, J J Optimization of control parameters for genetic algorithms In *IEEE Transactions on Systems, Man, and Cybernetics*, page 122 IEEE, SMC-16(1), January/February 1986 Vol 16(1)

[37] Grefenstette, J J Incorporating problem specific knowledge into genetic algorithms In Davis, L, editor, *Genetic Algorithms and Simulated Annealing*, chapter 4, pages 42–59 Morgan Kaufmann, Los Altos, CA, 1987

[38] Grefenstette, J J and Baker, J E How genetic algorithms work A critical look at implicit parallelism In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Mateo, CA, 1989 Morgan Kaufmann

[39] Holland, J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[40] Jones, D R and Beltramo, M A Solving partitioning problems with genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 442–449, San Mateo, CA, 1991 Morgan Kaufmann

[41] Jones, K Optimization of experimental data *International Laboratory*, pages 32–45, November 1986

[42] Kelly, J D and Davis, L Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 377, San Mateo, CA, 1991 Morgan Kaufmann

[43] Koza, J R *Genetic Programming On the Programming of Computers by Means of Natural Selection* MIT Press, Cambridge, MA, 1992

[44] Laszewski von, G Intelligent structural operators for the k-way graph partitioning problem In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52, San Mateo, CA, 1991 Morgan Kaufmann

[45] Levy, G C , Wang, S, Kumar, P , and Borer, P Multidimensional nuclear magnetic resonance spectroscopy and modeling of complex molecular structures A challenge to today's computer methods *Spectroscopy International*, 3(4) 22, 1991

[46] Li, T -H , Lucasius, C B , and Kateman, G Optimization of calibration data with the dynamic genetic algorithm *Analytica Chimica Acta*, 268(1) 123–134, 1992

[47] Liepins, G E and Vose, M D Representational issues in genetic optimization *Journal of Experimental and Theoretical Artificial Intelligence*, 2 101, 1990

[48] Lucasius, C B , Beckers, M L M , and Kateman, G Genetic algorithms in wavelength selection A comparative study *Analytica Chimica Acta*, 1993 Submitted

[49] Lucasius, C B , Blommers, M J J , Buydens, L M C , and Kateman, G A genetic algorithm for conformational analysis of DNA In Davis, L , editor, *Handbook of Genetic Algorithms*, pages 251–281, New York, NY, 1991 Van Nostrand Reinhold Chapter 18

[50] Lucasius, C B , Blommers, M J J , Werten, S , van Aert, A H J M , and Kateman, G Conformational analysis of DNA using genetic algorithms In Schwefel, H -P and Manner, R , editors, *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, pages 90–97, Berlin, 1991 Springer-Verlag Lecture Notes in Computer Science 496

[51] Lucasius, C B , Dane, A D , and Kateman, G On k medoid clustering of large data sets with the aid of a genetic algorithm Background, feasibility and comparison *Analytica Chimica Acta*, 1993 In press

[52] Lucasius, C B and Kateman, G Genetic algorithms for large-scale optimization in chemometrics An application *Trends in Analytical Chemistry*, 10 254–261, 1991

[53] Lucasius, C B and Kateman, G Towards solving subset selection problems with the aid of the genetic algorithm In Manner, R and Manderick, B , editors, *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, pages 239–247, Amsterdam, 1992 Elsevier

[54] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 1

General perspective *Computers & Chemistry*, 1993 Submitted

[55] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 2 Toolbox description *Computers & Chemistry*, 1993 Submitted

[56] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 1 Concepts, properties and context *Chemometrics and Intelligent Laboratory Systems*, pages 1–33, 1993

[57] Lucasius, C B , Weijer de, A P , Buydens, L , and Kateman, G CFIT A genetic algorithm for survival of the fitting *Chemometrics and Intelligent Laboratory Systems*, 19 337–341, 1993

[58] Manderick, B , Weger de, M , and Spiessens, P Genetic algorithms and the structure of the fitness landscape In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150, San Mateo, CA, 1991 Morgan Kaufmann

[59] Massart, D L , Vandeginste, B G M , Deming, S N , Michotte, Y , and Kaufman, L *Chemometrics A textbook* Elsevier, Amsterdam, 1988

[60] Mathias, K and Whitley, D Remapping hyperspace during genetic search Canonical delta folding In Whitley, D , editor, *Proceedings of the Second Workshop on the Foundations of Genetic Algorithms*, pages 167–186, San Mateo, CA, 1993 Morgan Kaufmann

[61] Michalewicz, Z *Genetic Algorithms + Data Structures = Evolution Programs* Artificial Intelligence Series Springer-Verlag, Berlin, 1992

[62] Michalewicz, Z and Janikow, C Z Handling constraints in genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 151, San Mateo, CA, 1991 Morgan Kaufmann

[63] Newell, A Heuristic programming ill-structured problems In Arnofsky, J , editor, *Progress in Operations Research*, pages 363–414, New York, NY, 1969 Wiley

[64] Oliver, I M , Smith, D J , and Holland, J R C A study of permutation crossover operators on the traveling salesman problem In Grefenstette, J J , editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 224, Hillsdale, NJ, 1987 Lawrence Erlbaum Associates

[65] Pao, Y H *Adaptive Pattern Recognition and Neural Networks* Addison-Wesley, Reading, MA, 1989

[66] Reed, J , Toombs, R , and Baricelli, N A Simulation of biological evolution and machine learning *Journal of Theoretical Biology*, 17 319–342, 1967

[67] Reeves, C R , editor *Modern Heuristic Techniques for Combinatorial Problems* Advanced Topics in Computer Science Blackwell Scientific Publications, Oxford, 1993

[68] Reingold, M R , Nievergelt, J , and Deo, N *Combinatorial Algorithms Theory and Practice* Prentice-Hall, Englewood Cliffs, NJ, 1977

[69] Rumelhart, D E , Feldman, J A, and Hayes, P J , editors *Parallel Distributed Processing* MIT Press, Cambridge, MA, 1986

[70] Schaffer, J D , Caruana, R A , Eshelman, L J , and Das, R A study of control parameters affecting online performance of genetic algorithms for function optimization In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 51, San Mateo, CA, 1989 Morgan Kaufmann

[71] Schraudolph, N N and Belew, R K Dynamic parameter encoding for genetic algorithms *Machine Learning*, 9(1) 9, 1992 Published in 1990 as Technical Report LA-UR-90-2795

[72] Schulze-Kremer, S Genetic algorithms for protein tertiary structure prediction In Manner, R and Manderick, B , editors, *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, pages 391–400, Amsterdam, 1992 Elsevier

[73] Siedlecki, W and Sklansky, J A note on genetic algorithms for large-scale feature selection *Pattern Recognition Letters*, 10 335–347, 1989

[74] Spears, W M Crossover or mutation? In Whitley, D , editor, *Proceedings of the Second Workshop on the Foundations of Genetic Algorithms*, pages 221 237, San Mateo, CA, 1993 Morgan Kaufmann

[75] Spears, W M and De Jong, K A An analysis of multi-point crossover In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, page 301, San Mateo, CA, 1991 Morgan Kaufmann

[76] Spears, W M and De Jong, K A On the virtues of parameterized uniform crossover In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, CA, 1991 Morgan Kaufmann

[77] Starkweather, T , McDaniel, S , Mathias, K , Whitley, D , and Whitley, C A comparison of genetic sequencing operators In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth*

International Conference on Genetic Algorithms, pages 69–76, San Mateo, CA, 1991 Morgan Kaufmann

[78] Starkweather, T , Whitley, D , and Mathias, K Optimization using distributed genetic algorithms In Davis, D , editor, *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, pages 176–185, Berlin, 1991 Springer-Verlag Lecture Notes in Computer Science 496

[79] Stender, J , editor *Parallel Genetic Algorithms Theory and Applications* Frontiers in Artificial Intelligence and Applications IOS Press, Amsterdam, 1993

[80] Syswerda, G Uniform crossover in genetic algorithms In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 2, San Mateo, CA, 1989 Morgan Kaufmann

[81] Syswerda, G Schedule optimization using genetic algorithms In Davis, D , editor, *Handbook of Genetic Algorithms*, page 350, New York, NY, 1991 Van Nostrand Reinhold Chapter 21

[82] Syswerda, G A study of reproduction in generational and steady state genetic algorithms In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, page 94, San Mateo, CA, 1991 Morgan Kaufmann

[83] Tanese, R Distributed genetic algorithms In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 434, San Mateo, CA, 1989 Morgan Kaufmann

[84] Vose, M D Generalizing the notion of schema in genetic algorithms *Artificial Intelligence*, 50 385, 1991

[85] Vose, M D and Liepins, G E Schema disruption In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–242, San Mateo, CA, 1991 Morgan Kaufmann

[86] Weijer de, A P , Lucasius, C B , Buydens, L M C , Kateman, G , Heuvel, H M , and Mannee, H A new approach to curve-fitting using natural computation *Analytical Chemistry*, 1993 Submitted

[87] Whitley, D The GENITOR algorithm and selection pressure Why rank-based allocation of reproductive trials is best In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, CA, 1989 Morgan Kaufmann

[88] Whitley, D and Kauth, J GENITOR A different genetic algorithm In *Proceedings of the Rocky Mountains Conference on Artificial Intelligence*, page 118, Denver, CO, 1988

[89] Whitley, D , Mathias, K , and Fitzhorn, P Delta coding An iterative search strategy for genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 77, San Mateo, CA, 1991 Morgan Kaufmann

[90] Whitley, D and Starkweather, T GENITOR II A distributed genetic algorithm *Journal of Experimental and Theoretical Artificial Intelligence*, 2 189, 1990

[91] Whitley, D , Starkweather, T , and Fuquay, D Scheduling problems and traveling salesmen The genetic edge recombination operator In Schaffer, J D , editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, San Mateo, CA, 1989 Morgan Kaufmann

[92] Whitley, D , Starkweather, T , and Shaner, D The traveling salesman and sequence scheduling Quality solutions using genetic edge recombination In Davis, D , editor, *Handbook of Genetic Algorithms*, pages 350–372, New York, NY, 1991 Van Nostrand Reinhold Chapter 22

PART II

DEVELOPMENT

CHAPTER 3

GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 1. General Perspective

Contents

GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 1. General Perspective

Abstract

Genetic algorithms comprise a novel methodology that has proven to be powerful in approaching complex, large-scale optimization problems in a wide variety of sciences, including computational chemistry recently. However, as it turns out, up to now the exploitation of this power is not at all a straightforward matter for many potential practitioners, among which computational chemists. Both parts of this paper provide keys to this group of scientists that should enable them to open gates towards genetic algorithm applications on computers. After a general introduction, Part 1 presents a taxonomy for genetic algorithm software. The discussion highlights important properties of different kinds of genetic algorithm software, and proposes a strategy to the applied scientist who needs an executable application without first having to become an expert in genetic algorithm science. The material presented is largely based on our past experience, which includes the insights that we gained during the development and use of our software library GATES. Applications built with GATES are spread across various fields of computational chemistry. GATES is described in Part 2 of this paper.

1 Preface

The beginning of the 1990ies marks an upsurge in the application of genetic algorithm methodology in a wide variety of applied sciences, including computational chemistry. These developments have triggered the interest of many more applied scientists, which has apparently led to a high demand for both fundamental and, especially, practical knowledge. Much of this knowledge is already available (more or less) in the mainstream literature on the topic: for pointers to tutorials on genetic algorithms and their applications, the reader is referred to Chapters 1 and 2 in this thesis; the list of applications in computational chemistry is extended here with: Leardi *et al.*, 1992; Unger & Moult, 1993; Jones *et al.*, 1993; Handley, 1993.

On the other hand, however, comparatively little attention has thus far been paid to the strategics of obtaining genetic algorithm application programs on computers. Especially this kind of information is vital to scientists who wish to start applying genetic algorithms. Our paper aims at a contribution to fill this gap in know-how. In doing so, we realize that a scientific public broader than computational chemists alone is potentially addressed. However, our primary objective in publishing this paper is to reach the computational chemist as effectively as possible, which explains our choice for the present, specialized journal.

Related to the fact that genetic algorithms comprise a young and rapidly evolving science, some parts of this papers are subjective or speculative. Wherever that is the case, however, the information given is in reasonable agreement with issues put forward by leading genetic algorithm experts at international conferences (e.g. in panel discussions) and on the international electronic mailing list for genetic algorithms (GADISTR@AIC.NRL.NAVY.MIL), which we monitored during the past few years.

Finally, throughout this paper the adverb "genetic" is regularly used as a shorthand for "concerning genetic algorithms".

2 Introduction

This section briefly outlines the elementary background of genetic algorithms, insofar as needed to fulfil the purposes of this paper, including Part 2 (Lucasius & Kateman, 1993c).

2.1 Principle

All search procedures have in common that they essentially improve an initial estimate of the true

solution (e.g. a random guess) according to some search heuristic. Genetic algorithms are implicitly parallel search procedures that employ search heuristics inspired by natural evolution processes according to Darwin (1859). This methodology was pioneered by Holland (1975) in his milestone research on adaptation in natural and artificial systems. After a period of relative dormancy, the application of genetic methodology has only recently gained the great momentum pointed out above.

For a further introduction to genetic algorithms, the reader is referred to Chapters II and 1 in this thesis. (In the chapter at hand, references made to other parts in this thesis are assumed to imply the there used terminology and conventions, unless explicitly overridden.)

2.2 Problem representation

Traditionally, candidate solutions are represented as strings. For a detailed discussion on string representations for problem types denoted as NUM, SUB, and SEQ, the reader is referred to Chapter 2 (Sections 2, 4.7 and 5) in this thesis.

Since recently, the literature on genetic algorithms increasingly addresses the issue whether non-string representations of candidate solutions are more appropriate in some cases. In particular, datastructures more complex than strings – e.g. trees structures, two- or higher dimensional arrays, etc. (Michalewicz, 1992) – are proposed for certain problems as alternatives that should give access to more efficient exploration heuristics.

Notwithstanding these new developments, any kind of encoded candidate solution is hereafter referred to as "string" without loss of generality, because any datastructure (no matter how complex) is ultimately stored in a linear array of computer memory. A minor exception to this convention is made only for circular SEQ problems. For these problems, it makes sense to interpret candidate sequences as if they are circular, i.e. as if the first and the last element are connected. Candidate sequences of this kind are sometimes called Hamiltonian cycles in the literature, including Part 2 (Lucasius & Kateman, 1993c). In contrast, Hamiltonian paths are candidate sequences for non-circular SEQ problems.

2.3 Generic flowchart

A genetic algorithm may be considered any algorithm that fulfils a general, more or less informal description of the evolutionary problem solving concept that resides in the mainstream literature on the topic. By this loose definition, innumerable flowcharts legally represent genetic algorithms, in principle. In this paper, our version of mentioned general description is called the *generic flowchart* of genetic algorithms. It is presented and discussed in Chapter 2 (Section 3) in this thesis; its evolution cycle is graphically summarized in Figure 1; a practically important taxonomy of its constituent modules is given in Chapter 2 (Section 4.2) in this thesis.

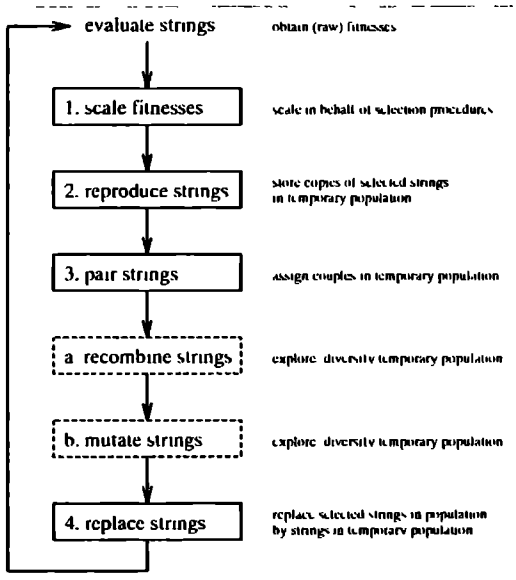

Figure 1: Generic evolution cycle in genetic algorithms: solid boxes and dashed boxes represent modules engaged with exploitation and exploration, respectively.

At any stage of a genetic run, the best candidate solution so far encountered is set aside, normally, and upon termination the best candidate solution ever encountered during the run is shown to the user.

2.4 Poor accessibility

In general, a genetic approach will be competitive only for a certain class of problems: complex, large-scale optimization problems, as a rule.

However, as it turns out, even when – or perhaps rather: especially when – this requirement is fulfilled, it is generally not straightforward to find a genetic configuration that features adequate (let alone optimal) performance. For this reason, genetic methodology may generally be deemed poorly accessible. Chapter 2 (Section 4.5) in this thesis sheds light on the causes of the poor accessibility of genetic methodology.

3 Notes on application scope

The collection of problems suitable for a genetic approach is vast and may be referred to as the overall application scope of genetic methodology, i.e. of the generic flowchart.

When routines are specified in order to (partially) instantiate the generic flowchart, the application scope is reduced, i.e. lies within the application scope of each of the constituent routines. An important kind of partial instantiation results in what is called a *genetic shell*: a flowchart wherein all routines, except the objective function (and the input routine that parameterizes it, module **–3**), are specified.

We have found it useful to categorize the application scope of routines comprising genetic algorithms, according to three hierarchical size classes: narrow, medium and wide; these correspond with strong, moderate and weak implicit assumptions about the target problem, respectively; for further details, the reader is referred to Chapter 2 (Section 4.6) in this thesis.

Exploration (recombination and mutation) routines with a medium application scope are called "domain-independent", notwithstanding the fact that they are actually confined to certain problem types, e.g. NUM, SUB, and SEQ.

4 Towards application

A decision whether or not to apply genetic methodology depends critically on how the properties of the methodology are, altogether, rated by the potential practitioner concerned. For most practitioners, the attractive properties (e.g. intuitive appeal, versatility, and search efficiency) outweigh the unattractive properties (e.g. poor accessibility), as may be concluded from the aforementioned recent upsurge in the application of genetic methodology. Accordingly, a great demand for genetic software has emerged, which, in turn, has provided the impetus for the production thereof (Navy Center for Applied Research in Artificial Intelligence, 1992; Ribeiro Filho *et al.*, 1993; Lucasius & Kateman, 1993a).

4.1 Software taxonomy

Distinct categories of genetic software, targeted to practitioners with different objectives, appear to have emerged. We identify the following main categories:

1. Education software
2. Application software
3. Development software

This taxonomy is elaborated on in Section 5.

4.2 Flavors of practice

In the remainder of this paper, special attention is paid to issues of interest to a distinct group of (potential) practitioners: experts in some real-world problem domain who pursue solving a complex problem within that domain (e.g. a molecular conformation problem), but who lack the necessary experience with genetic methodology (which, they believe, can offer good solutions). Individuals of this kind are hereafter referred to as "applied scientists" for brevity.

In contrast, and again for brevity, "genetic scientists" are hereafter understood to be specialists in genetic methodology and related programming, i.e. individuals who know how to avoid the common pitfalls that typically occur if the methodology would be applied naively. Moreover, they are understood to be mainly interested in the foundations of genetic algorithms rather than in real-world problem solving. This profile seems to fit the majority of scientists who have up to now contributed to the science of genetic algorithms. Rather than to embark on the laborious task to acquire the specialistic background knowledge of some real-world problem in order to obtain an objective function, genetic scientists generally prefer to use "realistic artificial" objective functions, which are relatively easy to design, yet have fitness landscapes of presumed real-world complexity and thus make real-world sense indirectly.

Up to now, a persistent dichotomy has prevailed between applied scientists and genetic scientists. This may be ascribed to the fact that the development of a genetic application program for solving a real-world problem, demands the integration of highly specialistic knowledge on two widely separated, extensive scientific areas, namely: the problem domain of interest (mainly mastered by the applied scientist) and genetic methodology (mainly mastered by the genetic scientist); in addition, a good command of computer programming is required.

The overall programming effort needed to obtain a genetic application program, can be substantially reduced by employing development software. However, practice learns that the skills needed at handling such software are generally mastered only by a limited group of individuals, namely: genetic scientists, and those applied scientists who happen to possess a fair amount of programming experience combined with elementary genetic knowledge. Individuals who command these skills will probably play a key role in the future spread of genetic application software among applied scientists, as will be elucidated later on; they are hereafter referred to as "developers" for brevity.

It is important to realize that the development of a genetic application program encompasses more than programming alone: the specific routines designed or chosen to construct the program must be parameterized; that is, meaningful values or value ranges must be specified for the genetic control parameters.

4.3 Strategy for applied scientists

Given that publicly available, problem-specific genetic application programs are still extremely scarce today, and that most applied scientists who pursue such a program are not developers, little choice seems to be left to most applied scientists than to recourse to a disciplined approach – e.g. an intelligently organized strategy – to achieve success. Such a strategy aims to minimize the common pitfalls related to the poor accessibility of genetic methodology. Based on our past experience in development and application, we recommend the next general strategy, comprising consecutively:

1. Acquire an elementary comprehension of genetic methodology; e.g. through textbooks and/or tutorial articles;
2. Acquire an overview of genetic software categories; e.g. through the next section of this article;
3. Acquire elementary skills at applying genetic methodology; e.g. through education software;
4. Seek sources of advanced genetic knowledge; e.g. genetic scientists and/or development software;
5. Develop the desired application program by borrowing (without learning) the necessary pieces of advanced genetic knowledge in 4; the ability to manipulate such knowledge is granted by the elementary knowledge acquired in 1, 2 and 3.

Steps 1, 2 and 3 are considered important for any novice practitioner; steps 4 and 5 maximize the effect/investment ratio. The applied scientist can realize step 5 by consulting a developer, as elucidated in Section 6 below.

5 Discussion of software taxonomy

This section treats the above-proposed taxonomy for genetic software in some detail. The order in which the material is presented, loosely indicates an increasing level of prerequisite acquaintance with genetic methodology expected from the user of the software concerned.

A roughly equivalent taxonomy of genetic software is independently given by Ribeiro Filho *et al.* (1993). In each of their software categories, an extensive overview of currently available software is provided; this overview is also available from the Navy Center for Applied Research in Artificial Intelligence (1992).

5.1 Education software

Genetic education programs are aimed at the novice practitioner who needs a gentle, comprehensive introduction to genetic methodology.

Related to the recent rise of interest in the application of genetic methodology, and the current

scarcity of genetic application programs nonetheless, it may be expected (especially during the next few years) that genetic education software will play a crucial role in the recognition and adherence of the methodology by applied scientists in research and business. However, for a truly major impact, it is required that genetic education software is available cheaply (or even freely), and is targeted to popular computer platforms, e.g. PCs. These circumstances may be difficult to achieve, since it is reasonable to expect that potential developers of education software (genetic scientists) will gain only little, financially. Moreover, the user has good reason to really insist on cheap genetic education programs, as the latter are not intended for long-term use, i.e. the user will soon reach a point in time after which little more can be learned.

Two categories of genetic education programs are considered next: (1) mechanics-oriented programs and (2) configuration-oriented programs.

5.1.1 Mechanics-oriented software

Mechanics-oriented genetic education programs are demonstration-based tutoring programs that show the user, through graphical animation, the workings of specific parts or aspects of a particular genetic application. These include, for instance, genetic operations on the population (recombination, mutation, selective reproduction, etc.), the building block principle (in recombination), exponential search behavior (due to the schema theorem), etc. The user interface also provides control over the speed of the genetic operations: "normal speed" (for an impression of the speed at which genetic algorithms normally solve problems), "slow motion" (a deliberate adaptation to the speed of the human eye) and "pause" (to allow intermezzos). Much room for configuration is, purposefully, not provided, i.e. fixed genetic operators are used and their parameterization is generally not encouraged.

An example of a mechanics-oriented genetic education program is Visual Genetic Algorithm (Melssen *et al.*, 1991), which runs under UNIX on SUN Sparc workstations.

5.1.2 Configuration-oriented software

Configuration-oriented genetic education programs are try-it-yourself tutoring programs that interactively allow the user to select genetic operators and objective functions from a fixed package of routines, and to parameterize them. The progress of the optimization process is graphically displayed on screen in order to provide feedback to the user, such as to allow adjustment of the choices made. In this way, the user gradually learns the kind of experience required in order to successfully handle more advanced genetic software (application software and development software), discussed below.

Upon start-up, the user enters a procedure for interactive, run-time configuration; this part of the user interface is menu-driven in order to avoid unnecessary complications. After the user has selected the desired routines with which a legal flowchart can be built, he/she is prompted for their parameterization; thereby, the program proposes realistic value ranges for the control parameters concerned (e.g. a mutation rate between 0.001 and 0.1). Next, the genetic algorithm thus obtained is invoked.

The objective functions from which the user can choose are, purposefully, computationally sparing; this is to avoid annoyingly long running times. The problem domains related to such objective functions are most often artificial, i.e. they concern "toy problems".

Objective functions for one- or two-dimensional NUM problems are educationally attractive, as these allow graphical display of the related fitness landscape in the decoded search space. Candidate solutions in the population are also displayed – as a swarm of points on the fitness landscape – in order to animate the optimization process; the currently best candidate solution and the best-so-far candidate solution are normally displayed in a distinct way.

For high-dimensional NUM problems, the fitness landscape can not be displayed in a straightforward way. For combinatorial problems, too, the fitness landscape can not be displayed in a straightforward manner. In both cases, only the currently best and/or best-so-far candidate solutions are displayed (in decoded form).

A distinctive part of the progress monitor is the performance monitor. It graphically displays the fitness distribution across the current population. The currently best- and average fitness in the population, as well as the best-so-far fitness, are displayed for the consecutive generations. These

statistics (and other diagnostic information) are optionally written or appended to special log files (report files) at regular intervals in the course of a run, for later (off-line) scrutiny.

An example of a configuration-oriented genetic education program is Genetic Algorithm Workbench (Hughes, 1992), which runs under DOS on IBM (and compatible) PCs, is mouse-driven, and requires EGA video display or emulation.

5.2 Application software

Genetic application programs are aimed at either genetic scientists (engaged in genetic research) or applied scientists (engaged in applied research or business). The further discussion primarily takes the interests of the latter group into account.

Genetic application software is targeted towards real-world problems of interest to applied scientists. Configurational lattitude is deliberately limited to what is stricktly needed, enabling the practitioner to obtain results with merely a basic understanding of genetic mechanics. We expect that future developments in genetic application software will critically determine the survival of genetic methodology. We base this prognosis on an analogy with real-world applications of expert systems and artificial neural networks – two other methodologies in the realm of intelligent problem solving, but with an earlier record of upsurge.

Targeted to only one problem (or at most to a few different, related problems), a genetic application program has a narrow application scope. Nonetheless, it can be run for many, if not countless, instances of the target problem – versions of the problem differing only in the domain-dependent input data used. This is important for the user, because it implies that routine usage of the program may ultimately pay back the prior investment in its development or purchase (provided, of course, that performance is acceptable, on average).

A genetic application program comes as an executable flowchart that comprises a fixed selection of routines. Configuration is limited to parameterization within comparatively narrow value ranges; this is to prevent accidental specification of unrealistic values (such as a mutation rate of 0.9, say). The objective function is usually computationally intensive, as is typical for real-world problems. For this reason, genetic application programs are most often aimed at fast computers, e.g. parallel (networks of) computers; it may be expected that decreasing hardware prices will accelerate these developments. Nonetheless, the running times are usually not on the scale of minutes, but rather hours, days, weeks, or even months. Under such circumstances, the practitioner is most often not particularly interested to sit by and closely monitor the evolution process. For this reason, genetic application programs are typically run in batch-like mode, implying that a rudimentary user interface generally suffices: most of the user's attention is asked at the start and at the end of a run, and on-line graphical display of progress is exceptional.

The input data, read at the start of a run (by modules **−3** and **−2** in the flowchart), serve to parameterize the objective function (i.e. to define the problem instance concerned) and the other, genetic, routines that comprise the flowchart. When not too abundant, these data can be read from the keyboard controlled by the user; otherwise they are read from a file of which the name is specified by the user at a prompt or on the command-line.

The genetic application program also produces output. At the end of a run, the best-so-far candidate solution is written to file (in decoded form) as "the solution" of the problem. Optionally, performance information and other diagnostics are written or appended to special log files during the course of a run. Furthermore, at regular intervals, the population may be written to a separate file as well; it can be loaded to continue a previously aborted session, e.g. as a provision against casual system failures such as a program crash (caused, for instance, by limited memory) or a general power loss.

In general, it is imperative that the user is knowledgeable about the problem domain concerned so that both the program's input and output can be handled properly.

By far most genetic application programs for real-world problems known up to now – including those for computational-chemical domains – have been developed in academia, where they have mainly found in-house use. Since recently, however, business-oriented applied scientists have started to develop application programs for commercial distribution – e.g. OMEGA, targeted to financial predictive modeling problems (Barrow, 1992). It is to be expected that application

programs geared to computational-chemical problems, too, will soon become commercially available, as sufficiently large interest groups appear to exist. It seems obvious that the commercialization of genetic application software in general, will turn out to be an important prerequisite for the further spread of genetic methodology within different sciences.

5.3 Development software

Genetic development software is aimed at reducing the programming effort on the part of the developer of genetic application programs. A genetic development package comprises, inter alia, a number of prefabricated, mainly domain-independent, routines which the developer can instantly use as parts of the aimed application program. However, development packages can, of course, not possibly anticipate on all parts of an application; the missing domain-dependent parts, among which always the objective function, need to be programmed and added by the developer.

Two categories of development software are considered next: (1) prototypes and (2) toolboxes. (In this discussion, the implementation of the objective function is understood to imply the input routine that parameterizes it, module **−3** in the flowchart).

5.3.1 Prototypes

A prototype is either an existing genetic application program in which it is intended to replace the objective function by another one, with the intent to obtain a new application program, or it is a genetic shell in which it is intended to insert the objective function, for the same purpose; the distinction between either developmental base is not drawn sharply.

Other parts of the prototype may also be replaced if the developer feels that is necessary or desirable. Examples of parts that do not stricktly need to be replaced are exploitation routines; they owe that to their wide application scope. Whether exploration routines need to be replaced when developing an application, depends, of course, on whether their application scope includes the target problem. In general, this is more likely to be the case when prototypes incorporate domain-independent exploration routines (e.g. for problem types NUM, SUB or SEQ) rather than domain-dependent exploration routines; the latter have a narrow application scope, whereas the former not. Discarding domain-dependent exploration routines for the moment, developers will generally be inclined to use prototypes comprising domain-independent exploration routines that are compatible with the target problem, so that only a new objective function needs to be supplied for the development of each application program; for example, if the target problem is of type NUM, say, then the developer will try to avoid the use of SUB and SEQ prototypes.

Depending on the producer of a prototype, either only the object code is made available or the source code as well. In the former case, the freedom of the developer is confined to supplying the objective function; in the latter case, all other routines can be manipulated (replaced, customized, etc.) as well.

Probably the most widely distributed prototype is GENESIS (GENEtic Search Implementation System), written by Grefenstette, in C (Davis, 1991); it is primarily aimed at NUM problem solving. In some cases, prototypes are also available as a part of a toolbox (see next section). An example is the toolbox OOGA (Object Oriented Genetic Algorithm), written by Davis (1991), in Common Lisp; it contains NUM- and SEQ prototypes. Another example is the toolbox GATES (Genetic Algorithm Toolbox for Evolutionary Search), written by Lucasius & Kateman (1993c), in C; it contains NUM-, SEQ- and SUB prototypes.

5.3.2 Toolboxes

For the developer who pursues the creation of a wide variety of genetic application programs on a regular basis, more configurational flexibility is usually needed than that offered by a prototype; prototypes are limited in that they are, to a large extent, preconfigured.

Suitable software for versatile development is embodied in a toolbox – an object-coded arsenal, or library, of prefabricated genetic routines, comprising numerous routines for fitness scaling, string reproduction, string pairing, string modification (recombination and mutation for several types of problems, e.g. NUM, SUB, and SEQ), string replacement; also, lower level routines, e.g. for random number generation and string decod-

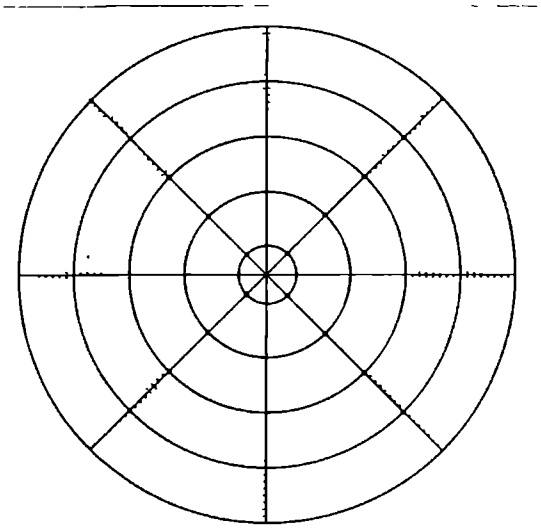

Figure 2: Idea behind the organization of routines in a toolbox: pie pieces represent lateral (modular) structure; concentric layers represent hierarchical structure. Documentation is usually limited to the outmost layers (shaded area) containing the highest level routines.

ing, are part of the toolbox. Depending on the producer of the toolbox, the source code of the routines may come along with the object code in order to provide even more flexibility to the developer.

The wide application scope of a toolbox may be appreciated by considering the large number of combinations of routines that result in legal prototypes (genetic shells).

It is important that the developer verifies that the toolbox he/she is planning to use is organized in such a way that: it is easy to develop applications; it can be accessed easily for maintenance; it can be extended easily (open architecture), e.g. with routines for unprecedented situations such as new problem types, or with prototypes that the developer has created from the toolbox and intends to use regularly as a developmental base, etc. This kind of organization demands an object-oriented approach, i.e. requires that the routines are structured into logical "objects" – laterally and hierarchically speaking (Figure 2).

Lateral structure is achieved when the routines are organized into logical categories (software modules), e.g. categories for initiation, recombination, mutation, selection, etc.; recombination and mutation categories may be subcategorized into NUM, SUB, and SEQ.

Hierarchical structure is achieved when a separate routine is available for each logical task and many logical tasks are subtasks of other logical tasks; for instance, a routine which recombines all assigned pairs of strings in the population, iteratively calls a routine which recombines one pair separately. Thus, the routines are layered in shells such that outer shells hide procedural details of inner shells; the inmost shell accommodates routines that are engaged most directly in data access.

The user of the toolbox may use routines from different levels when developing the application. In general, routines in inner shells – i.e. primitive routines – provide more control (flexibility), but also imply a larger programming effort. The outmost shell of a toolbox normally accommodates a number of prototypes, which minimize the programming effort to merely defining and linking the objective function in order to obtain the desired application program; examples of toolboxes with that feature are abovementioned OOGA and GATES.

Good documentation on all, or at least the most frequently selected, routines in a toolbox is indispensible. The producer of the toolbox may be inclined to take advantage of the hierarchical (layered) structure by skipping the documentation of routines in the inmost shells (Figure 1), anticipating that most developers will not want to access these routines explicitly anyway. As a result, a dramatic reduction in the amount of documentation can be obtained, saving the producer much work, and providing the user (the developer) a conceptually simpler overview with little loss of configurational freedom. The documentation comprises prosaic descriptions (comment on routines) and algorithmic examples (usage of routines) – either in the source code (if available) or in a hard-copied manual, or both. Prototypes, as they are at the top of the hierarchy, may themselves provide substantial documentation; this can be accomplished by accurately tracing the tree of implied function calls through the source code.

Automated development. A toolbox may come with a special utility, a front-end, that automates to a large extent the selection and integration of toolbox routines into a genetic application

program according to the developer's wishes.

In its simplest form, such a front-end is an interpreter that parses some high-level, genetic-dedicated command language. Additionally, the interpreter may include provisions (e.g. another special language) that facilitate the writing of objective functions.

A more advanced front-end is obtained when the interpreter is embedded in an interactive progam that incorporates a heuristic decision tree. The user traverses a chain of decisions through this tree as he/she answers asked questions, which may be accompanied by "intelligent" suggestions for an answer; the decisions are then cast into commands that are subsequently passed to the interpreter. In turn, the interactive progam may be embedded in a menu-driven, windows-based environment for even more convenience.

Examples of development software comprising a toolbox furnished with a front-end for automated development are EnGENEer (Robbins, 1993) and Splicer (Bayer & Wang, 1991).

6 Development through consultancy

This section sheds some light on how the applied scientist who has acquired elementary genetic knowledge, can develop his/her specific application program by consulting a developer. In doing so, the genetic development software usually used by the developer, comes within his/her reach for indirect usage.

Co-operatively, the applied scientist, or "client", and the developer, or "consultant", make a development plan, whereby distinct tasks are identified and then scheduled. Some tasks can be worked out independently by a single party; these are assigned according to closest match of personal skills. Ultimately, all separate pieces of produce are integrated in order to obtain the aimed application program.

The success of a development strategy based on consultancy, evidently depends in great part on the quality of the communication between client and consultant. In order to ensure a good communication, the client, as the dependent party, must: (1) possess some minimal amount of elementary genetic knowledge in order to at least globally understand what the developer is talking about; (2) have the skill at transforming and passing to the developer the required domain knowledge (concerning domain-dependent input/output facilities, an objective function, and, if appropriate, domain-dependent exploration routines) in a predigested form; that is, in a form closer to an algorithmic structure, or even, ideally, as instant routines. In order to facilitate the communication, a mediating knowledge engineer may be useful; a similar strategy is often followed in expert system development.

Development strategies based on consultancy seem to constitute a promising avenue for the applied scientist to obtain high-quality genetic applications with minimal (although certainly not little) effort. This is corroborated by the recent advent of developers operating on a commercial consultancy basis as roughly outlined above. With a significant proportion of their customers coming from academia and R&D in industry, the potential impact on scientific research in general should be evident.

7 Conclusions and outlook

Genetic algorithms comprise a powerful methodology for complex, large-scale optimization, but a proportional price must be paid in order to unveil this power and take advantage of it. That scientists are willing to pay this price, follows from the recent boom in reported applications of the methodology within numerous applied sciences, including computational chemistry recently.

The survival of applied genetic methodology depends largely on the future dissemination of genetic application software among significant interest groups. This, in turn, depends in great part on whether application software can be developed in a cost-effective way. A general strategy to that end has been recommended to the applied scientist on the basis of our past experience in development and application; it was laid out along the lines of a general taxonomy proposed for genetic software.

Once a significant spread of genetic application software has been attained, it may be expected that parallel- and hybrid applications (Lucasius & Kateman, 1993b) are going to play an increasingly important role next.

A newly emerging branch within the science of genetic algorithms – known as *genetic programming* (Koza, 1991; Koza, 1992; Angeline, 1993)

– has not been treated in this paper, but should be mentioned as an apparently promising future development in that it potentially bypasses some of the bottlenecks diagnosed for the application of traditional genetic methodology The idea behind genetic programming is that a population of computer programs evolves with the intent to solve problems Each program is basically a compound function, that is, a function obtained by combining a number of simple functional elements, selected from a predefined source set, into a tree The fitness of the computer programs in the population is obtained by assessing how well a predefined learning target is accomplished Thus, the computer programs essentially act as objective functions that evaluate themselves as they evolve Advocates of genetic programming argue that, in contrast with traditional genetic applications, the approach allows one to solve complex problems without the need to implement specialistic domain knowledge in the objective function – thus potentially obviating, inter alia, consultancy-based development strategies Whether genetic programming can be an asset to real-world problem solving is a question still to be answered, though

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007 We wish to thank Dr L Davis and Drs A P de Weijer for their critical comments and useful suggestions on the draft version of this paper

References

[1] Angeline, P J and Pollack, J B Competitive environments evolve better solutions for complex tasks In Goldberg, D E and Schaffer, J D , editors, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993 Morgan Kaufmann To appear

[2] Barrow, D OMEGA KiQ Ltd , Essex, United Kingdom, 1992

[3] Bayer, S E and Wang, L Problem solving with genetic algorithms and Splicer Technical Report AIAA-91-3836-CP, American Institute of Aeronautics and Astronautics, 1991

[4] Darwin, C *The Origin of Species by Means of Natural Selection the Preservation of Favoured Races in the Struggle for Life* John Murray, London, 1859 First edition text, edited with an introduction by J W Burrow, published in Pinguin Books 1968

[5] Davis, L , editor *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York, NY, 1991

[6] Handley, S Automated learning of a detector for alpha helices in protein sequences via genetic progamming In Goldberg, D E and Schaffer, J D , editors, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993 Morgan Kaufmann To appear

[7] Holland, J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[8] Hughes, M Genetic Algorithm Workbench Cambridge Consultants Ltd , Cambridge, United Kingdom, 1992

[9] Jones, G , Brown, R D , Clark, D E , Willett, P , and Glen R C Searching databases of two-dimensional and three dimensional chemical structures using genetic algorithms In Goldberg, D E and Schaffer, J D , editors, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993 Morgan Kaufmann To appear

[10] Koza, J R Concept formation and decision tree induction using the genetic programming paradigm In Schwefel H P and Manner, R , editors, *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, pages 124–128, Berlin, 1991 Springer-Verlag Lecture Notes in Computer Science 496

[11] Koza, J R *Genetic Programming On the Programming of Computers by Means of Natural Selection* MIT Press, Cambridge, MA, 1992

[12] Leardi, R , Boggia, R , and Terrile, M Genetic algorithms as a strategy for feature selection *Journal of Chemometrics*, 6 267–281, 1992

[13] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 1 Concepts, properties and context *Chemometrics and Intelligent Laboratory Systems*, 19 1–33, 1993a

[14] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 2 Representation, configuration and hybridization *Chemometrics and Intelligent Laboratory Systems*, 1993b In press

[15] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 2 Toolbox description *Computers & Chemistry*, 1993c Submitted

[16] Melssen, W J , Lucasius, C B , and Kateman, G VGA Visual Genetic Algorithm, Mechanics-Oriented Genetic Education Software, Laboratory for Analytical Chemistry, Katholieke Universiteit Nijmegen, the Netherlands, April 1991

[17] Michalewicz, Z and Janikow, C Z Handling constraints in genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 151, San Mateo, CA, 1991 Morgan Kaufmann

[18] Navy Center for Applied Research in Artificial Intelligence Information on commercial and public domain software related to genetic algorithms Public FTP archive FTP AIC NRL NAVY MIL Naval Research Laboratory, Washington, DC, 1992

[19] Ribeiro Filho, J L , Alippi, C , and Treleaven, P Genetic algorithm programming environments *IEEE Computer*, 1993 To appear

[20] Robbins, G EnGENEer – the evolution of solutions In *Proceedings of the Fifth Annual Seminar on Neural Networks and Genetic Algorithms*, 1993 To appear

[21] Unger, R and Moult, J A genetic algorithm for 3D protein folding simulations In Goldberg, D E and Schaffer, J D , editors, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993 Morgan Kaufmann To appear

CHAPTER 4

GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 2. Toolbox Description

Contents

GATES Towards Evolutionary Large-Scale Optimization: A Software-Oriented Approach to Genetic Algorithms. Part 2. Toolbox Description

Abstract

Starting from Part 1, Part 2 of this paper globally describes the toolbox GATES, elucidating how concepts of genetic algorithm methodology can be implemented for application. The prototype for numerical parameter estimation is treated in most detail. Additionally, an auxiliary utility for configuration, based on explorative walks in the search space, is outlined.

1 Preface

During the past few years, we constructed the toolbox GATES – Genetic Algorithm Toolbox for Evolutionary Search, in the computer language C (ANSI standard). This project was motivated by our pressing need for versatile, efficient, and reliable genetic development software – software that would facilitate the development of various genetic applications – at a time wherein publicly available genetic development software was scarce, limited in scope, and featured questionable maintenance support. Additionally, we realized that the development and use of GATES would provide us with a more profound understanding of what is practically relevant and feasible; furthermore, this experience is considered useful for the validation of forthcoming professional genetic development software in general.

The global description of GATES given in this paper, starts with a discussion roughly along the lines of the generic flowchart referenced in Part 1 (Lucasius & Kateman, 1993c), or Chapter 3 (Section 2.3) in this thesis. Thereby, the following shorthands are used for the sake of brevity. A "NUM prototype" is understood to be the NUM prototype in GATES. Mutatis mutandis, the shorthands "SUB prototype" and "SEQ prototype" are used; the shorthand "standard prototype" denotes either a NUM-, SUB- or SEQ prototype. These conventions extend to "GA"s, "configuration"s, and "option"s: a "NUM GA", for instance, is understood to be a genetic application program developed from the NUM prototype; a "SUB configuration", for instance, is understood to be a particular configuration which defines a SUB GA, or the effort done (or to be done) to achieve that; and a "SEQ option", for instance, is understood to be a legal choice made (or to be made) by the user to partially accomplish SEQ configuration.

2 Preliminary actions

The actions that are required before entering the evolution cycle illustrated in Chapter 3 (Section 2.3) in this thesis, are performed by modules **−3**, **−2** and **−1** in abovementioned generic flowchart. (In the chapter at hand, references made to other parts in this thesis are assumed to imply the there used terminology and conventions, unless explicitly overridden.)

GATES does not include routines for the parameterization of objective functions (module **−3** in the flowchart), for the simple reason that the target problem is normally not known in advance.

The parameterization of genetic routines (module **−2** in the flowchart) can be performed by so-called *housekeep* functions. There are three housekeep functions in GATES – one for each of the three types of standard GAs; the housekeep function for NUM GAs is discussed in some detail in Section 6.2 below. When a housekeep function is called, it starts reading a standard configuration file. The name of this file is specified as a command-line argument upon starting the standard GA concerned. As the housekeep function reads the information in the configuration file, it assigns a number of variables and dynamically

allocates memory for all arrays needed. For instance, the memory that serves to accommodate the vector of fitness values, is allocated as soon as the number of strings comprising the population is known from the configuration file. The advantage of run-time (i.e. dynamic) memory management over compile-time memory management is that any desired (and, of course, legitimate) change in the demands for memory can be implemented by simply editing the configuration file, thus obviating recompilation of the source code.

For the initiation of strings (module **-1** in the flowchart), only three initiation routines are available: one for each of the three types of standard GAs. These routines make use of a Fibonacci VLP (very long period) pseudo-random number generator (Marsaglia, *et al.*, 1990; James, 1990). The primitive of this generator generates random 24-bit mantissas – fractional parts of a single-precision real number.

Incidentally, all other random routines, and probabilistic routines, in GATES (i.e. for recombination, mutation, selection, etc.) are also based upon the Fibonacci VLP pseudo-random number generator. Among its salient properties are: good randomness and homogeneity; warranted repeatability; very long period (2^{144} in GATES), which can be adjusted; high efficiency; ample portability.

3 Exploitation

Exploitation routines – modules **1** (fitness scaling), **2** (string reproduction), **3** (string pairing) and **4** (string replacement) in the flowchart – affect the survival rate of strings in the population.

3.1 Fitness scaling

This section briefly outlines fitness scaling functions available as standard options. First, a general introduction is given.

In a genetic algorithm, either the reproduction procedure or the replacement procedure, or both, use a vector of fitnesses as input; fitnesses are numerical values which, respectively, indicate the utility of the strings in the population. This vector is the result of performing a number of, possibly 0 (!), numerical transformation, or *scaling*, steps on a vector of so-called raw fitnesses. In turn, the vector of raw fitnesses is the result of evaluating all strings in the population by the objective function; this vector is denoted as:

$$\mathbf{f}_0 = (f_{0,1}, \cdots, f_{0,N})$$

where the subscript 0 reminds of the fact that the fitnesses are raw, i.e. not (yet) scaled; N is the number of strings in the population ($N \geq 2$, otherwise recombination can not be applied) – a control parameter.

In practice, fitness scaling is usually conducted in a sequence of simple steps (Lucasius & Kateman, 1993b). The fitness scaling function for the first step is denoted as ξ_1. When it is applied, to $\mathbf{f}_0$, a new fitness vector, $\mathbf{f}_1$, is obtained:

$$\mathbf{f}_1 = \xi_1(\mathbf{f}_0)$$

In many implementations, including standard GAs, ξ_1 involves **translation**, defined by the relation:

$$f_{1,i} = \xi_1(f_{0,i}) = f_{0,i} - (f_{0,min} - \varepsilon)$$

where $f_{0,min}$ is the smallest fitness in $\mathbf{f}_0$; ε is a control parameter ($\varepsilon \geq 0.0$), called "fitness offset". The purpose of translation is to affect fitness ratios, or *competition*, between strings; competition is maximal for $\varepsilon = 0.0$.

In general, the n^{th} fitness scaling step ($n \geq 1$) can be formally summarized as:

$$\mathbf{f}_n = \xi_n(\mathbf{f}_{n-1})$$

Whether fitness scaling is required depends on whether the reproduction procedure or the replacement procedure selects strings on the basis of fitness *values* or merely on the basis of fitness *ranks*. In the latter case, fitness scaling is not needed (although it will not do any harm), because for consistency reasons, fitness scaling must conserve fitness ranks; this is called the monotonicity condition (Lucasius & Kateman, 1993b). An example of the former case is a procedure that selects strings at rates proportional to their scaled fitness – also known as *roulette selection* (Section 3.2.1 below). Without fitness scaling, if the maximum unscaled fitness in the fitness vector $\mathbf{f}_0$ is much larger than the average fitness, it is likely that the search converges prematurely (i.e. into a local optimum), especially at an early stage. It is therefore important that the reproduction rates are stabilized, which is why fitness scaling is needed. A further requirement imposed by roulette selection

is that the fitness vector is normalized, as elucidated below.

One may distinguish between *analytical-* and *algorithmic* fitness scaling functions. In the former case, ξ_n is an analytical (mathematical) transfer function. Otherwise, ξ_n is an algorithmic procedure without a known analytical definition, e.g. a complex deterministic procedure (say, a procedure with many nested conditions in its definition) or a probabilistic procedure.

For more detail on the fitness scaling functions discussed next, and on other fitness scaling functions, the reader is referred to Chapter 2 (Section 6.1) in this thesis.

3.1.1 Linear scaling

Linear fitness scaling (Goldberg, 1989) defines a linear relationship between $f_{n,i}$ and $f_{n-1,i}$ in such a way that the average fitness remains invariant, say $\bar{f}$, under the transformation, while the ratio between the maximal scaled fitness and the average fitness, hence the competition between strings, is dynamically kept stable, say s, during a run:

$$\frac{f_{n,max}}{\bar{f}} = s$$

This target ratio, called sensitivity, is a control parameter. The transfer function ξ_n for linear fitness scaling is given by:

$$f_{n,i} = \xi_n(f_{n-1,i}) = \frac{\bar{f}}{f_{n-1,max} - \bar{f}} \cdot ((f_{n-1,max} - \bar{f}s) + (s-1) f_{n-1,i})$$

Note that according to this equation, negative $f_{n,i}$ values occur when $f_{n-1,min} < (s\bar{f} - f_{n-1,max})/(s-1)$. This raises a problem for a number of selection criteria, among which aforementioned roulette selection, since these assume non-negative $f_{n,i}$ values. In these cases, therefore, linear fitness scaling can only be used if it is extended with an exception handler that avoids negative $f_{n,i}$ values. (This adds an algorithmic dimension to the scaling function.) An example of such an exception handler is one that simply sets any incidentally occurring negative $f_{n,i}$ to 0.0; in this case we speak of *static linear scaling*. Another suitable exception handler for linear scaling is one that dynamically decreases the target s down to the largest value that would just prevent the smallest $f_{n,i}$ from becoming negative upon scaling; in this case we speak of *dynamic linear scaling*. Both static- and dynamic linear scaling, specifically, are standard options.

3.1.2 Sigmoid scaling

Sigmoid fitness scaling basically segregates the population in a worst performing part (strings with a low fitness) and a best performing part (strings with a high fitness). The transfer function ξ_n for sigmoid fitness scaling has an inflection point about the average unscaled fitness $\bar{f}_{n-1}$ and is given by:

$$f_{n,i} = \xi_n(f_{n-1,i}) = 1 + \tanh\left((s-1)\left(\frac{f_{n-1,i}}{\bar{f}_{n-1}} - 1\right)\right)$$

where the control parameter s ($s > 1.0$) is called the segregation degree. Competition between strings within either part is mild but not meaningless. Strings from different parts compete most heavily. This competition is fairly stable during a run, owing to the segregation. Like in linear scaling, sigmoid scaling imposes a lowbound and a highbound on the scaled fitness.

3.1.3 Normalization

Although normalization does not affect the competition between strings, it is nonetheless required for certain reproduction- and replacement procedures, as pointed out above. The normalization function, ξ_n, is defined by:

$$f_{n,i} = \xi_n(f_{n-1,i}) = \frac{f_{n-1,i}}{\sum_{i=1}^{N} f_{n-1,i}}$$

where N is the population size.

3.1.4 Cumulation

Cumulation is an algorithmic scaling function, defined by:

$$\begin{aligned} f_{n,1} &= f_{n-1,1} \\ f_{n,i} &= f_{n-1,i} + f_{n,i-1} \quad \text{for } i = 2, \cdots, N \end{aligned}$$

where N is the population size. When cumulation is applied to a normalized fitness vector, $\mathbf{f}_{n-1}$, then the scaled fitness vector obtained, $\mathbf{f}_n$, has $f_{n,N} = 1.0$.

3.2 String reproduction

It is convenient to view string reproduction (module **2** in the flowchart) as a special fitness scaling step – one that is appended to the preceding fitness scaling steps (module **1** in the flowchart). It is special in that it is always algorithmic, as the "scaling" function used is a string selection criterion, and it is always implicit, as the "scaled" fitnesses are not actually calculated and stored as a vector in memory, but, instead, manifest themselves actively as selection rates – or, equivalently, as reproduction rates – that can be observed.

Each reproduction procedure in GATES can be used in two modes, called "full reproduction" and "partial reproduction", respectively. In order to understand this distinction, recall that any reproduction procedure is applied according to the following general scheme. A selection criterion is applied N' times to a population of N strings by a reproduction procedure, where N' is a control parameter that satisfies $2 \leq N' \leq N$; each selected string is copied to a temporary population, which therefore has size N' after reproduction is completed. (The minimal value of N' is 2, because recombination can not be applied for a smaller N'.) Full reproduction refers to the case wherein $N' = N$, whereas partial reproduction refers to the case wherein $N' < N$.

Although GATES supports both full- and partial reproduction, standard GAs are confined to full reproduction – the traditional approach.

Each reproduction procedure in GATES is applied in "full return" mode, which means that no measures are taken to change the expected selection rate of a string each time after it is selected for reproduction. (We use the term "return" here, instead of "replacement" as is customary in statistical literature, in order to avoid confusion with genetic string replacement.)

"Partial return" is not supported by GATES, but it can be achieved with only small adjustments to the source code. In this mode, the expected selection rate of a string is purposefully decreased each time after it is selected for reproduction; in case that such a decrease would lead to a negative rate, it is simply set to zero in some way. (In the special case that the expected selection rate of a string becomes zero after its first selection, one speaks of "no return"; that is, "once selected, never selected again".) Evidently, partial return encourages population diversity.

The selection criteria in GATES are designed and organized in such a way that it is fairly easy to combine them sequentially into so-called *compound selection criteria*. A sequential compound selection criterion uses one selection criterion for the first N'_1 string selections, another selection criterion for the next N'_2 string selections, and so on if there are more constituent selection criteria, where $N'_1, N'_2, \cdots$ are control parameters that satisfy $N'_1 + N'_2 + \cdots = N'$ – the size of the temporary population. A counter is used to keep track of the number of string selections made; it is initiated to 0 when the reproduction procedure is started, and incremented by 1 upon each string selection. The counter's value determines which constituent selection criterion should be used at a particular time step.

Standard GAs use sequential compound selection: elitist selection followed by either roulette selection or rank-based threshold selection, all of which are discussed next. For more detail on these selection criteria, and on other selection criteria, the reader is referred to Chapter 2 (Section 6.2) in this thesis.

3.2.1 Roulette selection

In roulette selection (De Jong, 1975; Goldberg, 1989), a string is selected (reproduced) probabilistically with an expected rate proportional to its scaled fitness.

A reproduction procedure that uses the roulette selection criterion, takes as input a fitness vector that has first undergone normalization, and then cumulation; hence, the last element in the fitness vector equals 1.0. The need for this two-step fitness scaling, stems from the way the selection criterion is implemented.

Roulette selection is implemented as follows. First, a random real number in the range [0.0, 1.0) is generated; this number is called a *key*. Next, using binary search (Knuth, 1973), the fitness vector is searched for the smallest element that is not smaller than the key. Finally, the index of this element is returned; the string with that index is reproduced.

3.2.2 Rank-based threshold selection

In rank-based threshold selection (Lucasius & Kateman, 1993b), a string is selected randomly

among the N_{best} best performing strings, where N_{best} is an integer control parameter ($2 \leq N_{best} \leq N$). That is, strings with a fitness rank below the threshold $N - N_{best}$ stand no chance of being reproduced.

A reproduction procedure that uses the rank-based threshold selection criterion, takes as input a vector of string indices sorted according to fitness rank. This vector, denoted as **q** next, is obtained as follows. First, a so-called *identity permutation* is created. For instance, for $N = 10$ the identity permutation is the bottom row of integer values in:

index	0	1	2	3	4	5	6	7	8	9
value	0	1	2	3	4	5	6	7	8	9

(assuming indices start counting at 0). Next, a procedure is applied which quicksorts (Knuth, 1973) the fitness vector in descending order and, on the fly, co-sorts the identity permutation to obtain a new permutation; this permutation is the aimed **q**. For instance, if the initial fitness vector is the bottom row of real values in:

index	0	1	2	3	4	5	6	7	8	9
rank	3	7	6	0	4	9	5	1	8	2
value	0.06	0.14	0.12	0.00	0.08	0.18	0.10	0.02	0.16	0.04

then **q** is obtained as the bottom row of integer values in:

index	0	1	2	3	4	5	6	7	8	9
rank	9	8	7	6	5	4	3	2	1	0
value	5	8	1	2	6	4	0	9	7	3

Using **q**, the rank-based selection criterion generates a random integer i in the range $[0, N_{best})$ and returns q_i; the string with that index is reproduced (assuming indices start counting at 0).

3.2.3 Elitist selection

Upon successively applying the elitist selection criterion (Goldberg, 1989) in a reproductive session, first the best performing string is selected, then the next-to-best performing string is selected, and so on, until the N_1' best performing strings have been selected, where N_1' is an integer control parameter ($0 \leq N_1' \leq N - 2$). Entirely free of any stochasticity, elitist selection is generally qualified as deterministic.

Like a reproduction procedure based on rank-based threshold selection, a reproduction procedure that uses the elitist selection criterion, too, takes as input a vector of string indices sorted according to fitness rank; again, we denote this vector as **q**. The elitist selection criterion uses a counter, i, that is initiated to 0 when the reproduction procedure is started, and incremented by 1 in each of the N_1' steps. For each i, the criterion returns q_i; the string with that index is reproduced.

The elitist selection criterion makes applicable sense only when it is used in a sequential compound selection criterion; hence the control parameter N_1'. The compound selection can be instructed to skip elitist selection. To that end, the user must simply specify $N_1' = 0$.

3.3 String pairing

In preparation for recombination (module **a** in the flowchart), the N' strings in the temporary population are paired (module **3** in the flowchart); if N' is odd, then the last string gets no partner and will simply pass recombination unmodified. String pairs can be assigned according to various criteria. GATES is limited to a simple, traditionally used pairing criterion: random pairing. Some more advanced pairing criteria are reviewed in (Lucasius & Kateman, 1993b).

3.4 String replacement

String replacement proceeds along lines reminiscent of string reproduction: a string selection criterion is applied N' times to a population of N strings, where $2 \leq N' \leq N$; each selected string is replaced by a string withdrawn from the temporary population, which initially has size N'.

Any selection criterion that can be used for string reproduction can, in principle, also be used for string replacement, provided some inverting transformation is applied to the elements in the original raw fitness vector (e.g. simply sign inversion) before the normal fitness scaling procedure is performed. An inverting transformation causes reversal of the fitness ranks.

Each replacement procedure in GATES can be used in two modes, called "full replacement" ($N' = N$) and "partial replacement" mode ($N' < N$). Standard GAs are limited to full replacement, as they are limited to full reproduction, too. More

commonly, the latter operational mode is also referred to as generational population handling, or as replacement without generation gap.

Each replacement procedure in GATES is applied in "no return" mode. This means that as soon as a string is selected for replacement, its expected selection rate is set to zero in some way (definite replacement).

4 Exploration

This section briefly passes in review some well-established exploration operators, i.e. recombination- and mutation operators, that are available in GATES as standard options. Technical aspects of related (de)coding are elucidated. For further details concerning the exploration operators treated, and other exploration operators, the reader is referred to Chapter 2 (Section 5) in this thesis.

The following general remarks can be made for all exploration operators.

In order to achieve good search performance, genetic algorithms must be configured in such a way that important items of information, or properties, accumulated in the population during past generations, are maximally preserved (i.e. minimally lost) during the exploration – recombination in particular. This plausible principle is generally known as the *building block principle* (Goldberg, 1989), and is widely regarded as an important guideline in the design, choice, and analysis of efficient recombination operators for particular search tasks.

In the recombination procedure (module **a** in the flowchart), the recombination operator is iteratively applied to the string pairs assigned in the temporary population by the pairing procedure (module **4** in the flowchart). For each of the three standard prototypes, a different set of recombination operators exists. The desired recombination operator is selected and parameterized through the appropriate standard configuration file.

A control parameter common to all recombination operators is the recombination probability, p_r. When a particular recombination operator, say the C-function `RecombineStringPair()`, is applied to a string pair, then only with probability p_r that operator is actually applied. In GATES, this is accomplished by using a simple macro, defined (in the module with random utilities) as:

```
#define WITH_PROBABILITY(Probability) \
        if (UNI() < (Probability))
```

where `UNI()` is a function which generates a random number in the range [0.0, 1.0). For example, when $p_r = 0.8$, one iteration in the recombination procedure amounts to:

```
WITH_PROBABILITY(0.8) \
   RecombineStringPair();
```

In the mutation procedure (module **b** in the flowchart), the mutation operator is iteratively applied to the strings in the temporary population. For each of the three standard prototypes, a different set of mutation operators exists; the desired mutation operator is selected and parameterized through the appropriate standard configuration file.

A control parameter common to all mutation operators is n_m: the number of elementary mutations to perform in a string, e.g. the number of bit inversions if a binary string is concerned. The elementary mutations are imposed at random locations in the string concerned. In all mutation operators in GATES, n_m can be specified according to two basic modes: fixed or distributed. The former case speaks for itself. In the latter case, n_m is derived from p_m: the probability by which each of all, say L, possible elementary mutations in the string occurs; n_m is binomially distributed, with mean $p_m L$ and variance $p_m L(1 - p_m)$.

In the remainder, mnemonics are used as the names for recombination- and mutation operators. Thereby, the first letter in a name denotes the type of encoding that underlies the operator concerned: **B** for binary encoding (applies to operators in NUM GAs), **D** for direct subset encoding (applies to operators in SUB GAs), and **P** for permutation encoding (applies to operators in SEQ GAs). An example is **B_MX** (binary multipoint recombination), discussed in Section 4.1.2 below. ("X" in the names graphically suggests the crossover of two chromosomes – biological recombination.)

4.1 The NUM case

The NUM case is elaborated on in most detail. This is motivated by the fact that genetic algorithms for NUM problem solving have the longest

tradition of investigation and are attractive for beginners in that the heuristics of NUM exploration operators are relatively simple to understand. The discussion also serves the description of the NUM prototype in Section 6 below.

4.1.1 Representation

In order to facilitate the further discussion, the following conventions and terminology concerning the encoding of real values for numerical problem parameters are introduced.

In any encoding, the value of a numerical problem parameter can be represented as a finite-length *word* composed of characters, or *digits*, copied from some k-cardinal *alphabet* ($k \geq 2$ and k is finite). A contiguous concatenation of words forms a *string* that encodes a candidate solution; in a D-dimensional problem, D words are concatenated.

Different words in a string may have different lengths; the number of digits in the word for the i^{th} problem parameter, X_i, is denoted as ℓ_i ($\ell_i \geq 1$); the digitization resolution for X_i is $\rho_i = k^{\ell_i}$. The vector of word lengths for an arbitrary string in the population is called the *string format*: $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_D)$; all strings in a population have the same string format. In the special case that all word lengths are the same, the string format is referred to as "homogeneous". The sum of all word lengths is the string length, $L = \sum_{i=1}^{D} \ell_i$; note that $L = D$ when all $\ell_i = 1$.

For $k = 2$ (binary encoding) a word would logically be called a bitword. In this specific case, however, the term *bitfield* is preferred in order to maintain the term *bitword* as used in computer terminology: a "natural", i.e. addressable, unit of computer memory in a particular hardware platform. Among motives for the use of a binary encoding are: more implementational convenience and greater speed of execution can be achieved, as computers naturally manipulate information in binary form. In addition, there are theoretical advantages in many cases; these are explained in Chapters 1 and 2 in this thesis.

A contiguous concatenation of bitfields forms a *bitstring*. Bitstrings can be stored in different ways in computer memory, e.g. as a bytestring in which each byte (bitword) is a conceptual bit, i.e. purposefully restricted to the values 0 and 1. Although this is implementationally probably the most convenient approach, it is not very efficient in terms of processing time and memory utilization. For this reason, GATES enables the developer to store and manipulate *compact bitstrings* – bitstrings in which the bits correspond to "actual" computer bits. In practice, this leads to a considerable gain in memory utilization; the gain in processing time, however, is usually overshadowed by the computational burden of, inter alia, the objective function (especially in case of real-world problems). The gain in memory utilization is especially attractive when high-dimensional target problems are tackled. For instance, if the target problem is defined by 1000 problem parameters that are each encoded by 8 bits, then a population of 500 bitstrings will occupy only 500 kB memory – within memory limits of many contemporary small computers.

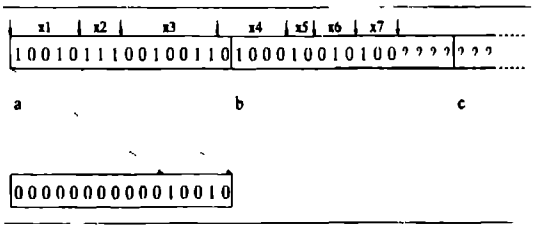

Figure 1: Top: storage of a compact bitstring; rectangles are 16-bit bitwords with addresses $a, b, \cdots$, respectively, whereas the ↓s delimit bitfields that correspond with the problem parameters $(X_1, \cdots, X_7)$; ?s denote unused bits. Bottom: read-access of a bitfield (for X_1), using an auxiliary bitword.

Figure 1 illustrates principles of storage and access of compact bitstrings. The depicted compact bitstring encodes the values of 7 problem parameters – $X_1, \cdots, X_7$ – using bitstring format $(5, 3, 7, 5, 2, 3, 3)$; hence the length of the bitstring is $L = 28$. The bitstring is stored left-aligned in two contiguously concatenated 16-bit bitwords, with addresses marked as a and b, respectively; by left-alignment we mean that the first bit of the bitstring coincides with the address of the first bitword, thus ensuring that the bitstring as a whole is addressable. The bits beyond the 28^{th} bit in the bitstring are unused. In general, in order to accomplish minimal memory utilization, one needs to allocate the minimal number of bitwords with a total bit count not smaller than the bit count of the accommodated bitstring; for the bitstring used

in the example, this means that the rightmost 4 bits in bitword b are wasted.

Since, in general, bitfields are not addressable (as they do not necessarily start where bitwords start), special access procedures are needed in order to retrieve or modify their integer interpretation. Bitfield retrieval -- performed by a special read-access procedure – is a preliminary step in decoding the value of a problem parameter, whereas bitfield modification – performed by a special write-access procedure – is a final step in recombination- and mutation operators for NUM GAs.

To understand the concept behind read-access, consider Figure 1 again. The 5-bit bitfield for problem parameter X_1 is copied right-aligned to an auxiliary bitword, and the unused leftmost bits in the bitword are set to 0; by right-alignment we mean that the least significant bits of the auxiliary bitword are used. In this way, the integer interpretation (value) of the auxiliary bitword corresponds with the integer interpretation of the bitfield concerned. This integer uniquely identifies a digital level in the predigitized, predefined real value range of the problem parameter concerned; for further details, the reader is referred to Chapter 2 (Section 5.2.1) in this thesis.

The value of the integer depends on how exactly the bitfield of interest was decoded. Apart from regular binary decoding, which corresponds with the way in which computers normally treat binary numbers, other forms of binary decoding exist, e.g. Gray binary decoding. The latter is preferred by many practitioners for reasons mentioned in (Caruana & Schaffer, 1988; Lucasius & Kateman, 1993b); for details on principles of Gray binary decoding, the reader is referred to Chapter 2 (Section 5.2.1) in this thesis.

Technically, Gray binary decoding proceeds as follows. The regular integer that corresponds with the bitfield of interest is regarded as a Gray integer, which should therefore be "deGrayed". This mapping results in a new regular integer, which is used instead of the original regular integer now to identify a digital level in the predigitized, predefined real value range of the problem parameter concerned. An efficient deGraying procedure is described in (Reingold *et al.*, 1977); in GATES it is defined as specified in Frame 1.

Like in read-access, in write-access, too, an auxiliary bitword is used. Incidentally, note that in either case the use of an auxiliary bitword for bitfield access implies that the maximal allowed size of a bitfield equals the size of a bitword. Accordingly, any bitfield resides within at most two bitwords; an example of a bitfield that resides within two bitwords is the bitfield for problem parameter X_4 in Figure 1.

```
int GrayToRegular(int Gray)
{
    int Regular = 0;

    while (Gray)
    {
        Regular ^= Gray;
        Gray >>= 1;
    }
    return Regular;
}
```

Frame 1: Function for deGraying.

The conventions for storage and access of compact bitstrings as described above, apply to GATES, except that 32-bit instead of 16-bit bitwords are used. The main reason for using 32-bit bitwords is the greater portability, as ANSI standard C guarantees that variables of the type `long int` are 32-bit bitwords, irrespective of the hardware platform used. Moreover, a 32-bit upperbound to the bitfield size of problem parameters is less restrictive than a 16-bit upperbound with regard to the digitization resolutions that can be attained.

Hereafter, the recombination- and mutation operators for the bitstrings are dichotomized according to whether the elementary modifications are targeted to the level of bits or to the level of bitfields. Bit-level operators generally feature a higher *exploratory power*, i.e. a larger number of points in the search space that are potentially reachable, on average, than their bitfield-level counterparts; more detail on this concept is provided in Chapter 2 (Section 5.2.1) in this thesis. The names used hereafter for any bit-level operator and its bitfield-level counterpart are identical, except that in the latter case a prime (') is appended to the name in order to make the distinction.

4.1.2 Recombination

The recombination operators among NUM options are: **B_MX**, **B_MX'**, **B_UX** and **B_UX'**.

B_MX and B_MX'. The properties of the **B_MX** operator – bit-level multipoint recombination, described in (Holland, 1975; De Jong, 1975) – were recently outlined in (Eshelman *et al.*, 1989; Spears & De Jong, 1991a). The operator is applied to two parent strings, with length L (in bits), as follows. First, one string is cut into segments at n_b different, randomly selected positions, or breakpoints ($1 \leq n_b \leq L-1$); the other string is cut at the same breakpoints. Then, the segments in each second pair of segments at the same position are swapped in order to obtain the two child strings. The entire operation is carried out with probability p_r, as pointed out above. Thus, the control parameters of **B MX** are n_b and p_r. In practice, frequently used values for n_b are 1 and 2.

The **B_MX'** operator is similar to **B_MX**, except that the swapped string elements are bitfields instead of bits; that is, the choice of the breakpoints is constrained such that these always fall between adjacent bitfields ($1 \leq n_b \leq D-1$).

B_UX and B UX'. The properties of the **B UX** operator – bit-level uniform recombination, recently introduced by Syswerda (1989) – were recently outlined by Eshelman *et al.* (1989) and Spears & De Jong (1991b). The operator is applied to two parent strings, with length L (in bits), as follows. First, n_s different bit positions are selected randomly ($1 \leq n_s \leq L$). Then, the bits in each pair of bits at the same selected position are swapped in order to obtain the two child strings. The entire operation is carried out with probability p_r. Thus, the control parameters of **B_UX** are n_s and p_r; the ratio n_s/L is called the bit swap rate, r_s. As can be easily verified, **B_UX** is symmetric in that $n_s = x$ and $n_s = L - x$, i.e. $r_s = y$ and $r_s = 1.0 - y$, are equivalent parameterizations; hence, maximal exploration of the search space is attained for $r_s = 0.5$. Incidentally, **B_UX** may be implemented alternatively such that r_s is distributed. Then, r_s is derived from p_s: the probability by which each of the L possible positionwise bit swaps between the strings occurs; r_s is binomially distributed, with mean p_s and variance $p_s(1-p_s)$.

The **B_UX'** operator is similar to **B UX**, except that the swapped string elements are bitfields instead of bits ($1 \leq n_s \leq D$).

Building block principle; positional bias. A distinguishing feature of **B_MX**, **B_MX'**, **B_UX**, and **B_UX'** is that the aforementioned building block principle must be implemented at the level of representation, as it is not already implemented heuristically; that is, their recombination heuristics are simple – not purposefully dedicated to any particular preservational task whatsoever.

How the building block principle can be implemented at the level of representation in **B_MX**, **B_MX'**, **B_UX**, and **B_UX'** is elucidated next.

In **B_MX** and **B_MX'** the building block principle is realized through the placement of the bitfields in the bitstrings. In particular, practitioners tend to place the bitfields for any given pair of problem parameters closeby in the bitstrings if it is known that these parameters are strongly correlated (in the sense that large fitness contributions occur only when both parameters adopt certain values simultaneously), because this maximizes the chance that both bitfields are propagated together from a parent string to a child string – hence that important information is preserved. Because of the effect of bitfield placement on the performance of **B MX** and **B_MX'**, these operators are said to exhibit *positional bias* (Eshelman *et al.*, 1989; Chapter 2 (Section 5.2.2) in this thesis). If the positional bias is taken advantage of as indicated above, i.e. when the bitfields in the bitstrings are placed in a so-called *connected sequence* (Frame 2), then the building block principle is best complied with.

In the NUM prototype, the placement of the bitfields in the bitstrings is accomplished by means of a *placement permutation* comprising D elements, such that for placement permutation $\mathbf{q}$, say, the i^{th} problem parameter is encoded as the q_i^{th} bitfield in a bitstring.

For certain target problems, however, it is difficult or even impossible to exploit the positional bias of **B_MX** or **B_MX'** (or any other positionally biased recombination operator for that matter). For instance, it may be the case that unknown correlations exist between the problem parameters. Then, the number of bitfield orders that one would have to consider to find the optimal order is in many cases prohibitively large (i.e. $D!$ for a D-dimensional target problem). Moreover, the proportion of correlated problem parameters may be so large that it is simply impossible to place the bitfields for each correlated pair closeby

Consider a bitstring that comprises D bitfields, with a total of L bits (In the special case that all bitfields comprise one bit. $D = L$) Let $r_{i,j}$ be the correlation between the bitfields x_i and x_j, how to determine $r_{i,j}$ is not important for now

The *causal problem complexity*, C, may be defined as $C = \prod_{i>j}(1 + \|r_{i,j}\|)$. The qualification "causal" derives from the fact that this definition aims to explain problem complexity in terms of correlations between problem parameters, the causal approach to problem complexity was introduced in Chapter 1 (Section 3.8 3) in this thesis. Note that C is minimal, 1, when all $r_{i,j} = 0$

Let $\delta_{i,j}$ be the distance by which x_i and x_j are separated in the bitstring according to an acceptable measure, e g the number of bits lying between the bitfields

The *δ-weighed causal problem complexity*, C', is now defined as $C' = \prod_{i>j}\frac{\delta_{i,j}}{L}(1 + \|r_{i,j}\|)$ For given $r_{i,j}$ values, C' can be minimized by manipulating the $\delta_{i,j}$ values ("weights") through bitfield placement. Any placement for which C' is minimal, is called a *connected sequence* of the bitfields the most strongly correlated bitfields are placed as near as possible to each other With bitfields placed in a connected sequence, the positionally biased recombination operators **B_MX** and **B_MX'** perform optimal, i e comply best with the building block principle

In pursuit of the $r_{i,j}$ values – which often comes down to a difficult task, especially in case of a real-world target problem – one should always start from the system on which the target problem is based An example of such a system is a macromolecule for which an optimal three-dimensional structure, or conformation, must be found. By definition, $r_{i,j}$ is equal to $R_{i,j}$ the correlation between problem parameters X_i and X_j that correspond to bitfields x_i and x_j, respectively Any information on $R_{i,j}$ – quantitative (ideally) or qualitative – facilitates the determination of a connected sequence of the bitfields In many cases, theoretically grounded knowledge about the system allows one to make at least qualitative statements about $R_{i,j}$. (With empirical information additionally available, quantitative statements can often be made as well)

For instance, consider the conformation problem Adopt as the problem parameters torsion angles along single chemical bonds in the macromolecule Let $d_{i,j}$ be the distance between the two single chemical bonds described by torsion angles X_i and X_j, respectively, $d_{i,j}$ may be defined in several ways, e g simply as the minimal number of separating chemical bonds Since correlations between the torsion angles are a result of spatially interacting molecular components, it is reasonable to assume, in first approximation, that $R_{i,j}$ is inversely proportional to $d_{i,j}$ in some way

In many other problems in which the interaction between the constituent components of the system depends on their separation in some space (including time, wavelength, etc), too, it is often not particularly difficult to make at least qualitative statements about correlation between problem parameters that describe different components, in many cases, the correlation is in some way inversely proportional to the separation This enables the practitioner to make educated guesses at a connected sequence of bitfields

Frame 2: Connected sequence of bitfields, explained in terms of causal problem complexity.

in a bitstring. Under these circumstances, practitioners will normally seek ways to apply less- or non-positionally biased recombination, since otherwise the positional bias is likely to encourage the search in the wrong direction. Thus, when the target problem is very complex, i.e. when by far most problem parameters are heavily correlated, the building block principle can be best complied with now only by purposefully avoiding positionally biased recombination.

The positional bias of **B_MX** and **B_MX'** can be decreased – but not eliminated – by increasing n_b. Minimal positional bias of **B MX** and **B MX'** is achieved for $n_b = L-1$ and $n_b = D-1$, respectively.

The positional bias of **B_UX** and **B UX'** is zero for any legal n_s (Spears & De Jong, 1991b; Chapter 2 (Section 5.2.2) in this thesis). These operators may be considered a better choice for the most complex NUM problems.

4.1.3 Mutation

The mutation operators among NUM options are **B_M** and **B_M'**.

The **B_M** operator – bit-level jump mutation – is applied to one string, with length L (in bits), as follows. First, n_m different bits are selected randomly ($1 \leq n_m \leq L$). Then, the values of these

bits are flipped. Thus, **B_M** is parameterized by n_m.

The **B_M'** operator – bitfield-level step mutation – is applied to one string, with length D (in bitfields), as follows. First, n_m different bitfields are selected randomly ($1 \leq n_m \leq D$). Then, the integer values of these bitfields are modified by performing integer steps, as follows.

For the i^{th} bitfield selected for mutation, the integer value, x_i, of the bitfield is copied (using an auxiliary bitword in a manner as illustrated in Figure 1) and an integer step, s_i, is calculated as described shortly. Then, with probability 0.5, x_i is increased, otherwise decreased, by s_i, subject to the condition that x_i is not changed if it would either overflow (i.e. become larger than $2^{\ell_i} - 1$, where ℓ_i is the bitfield length in bits) or underflow (i.e. become smaller than 0). Finally, the bitfield is modified according to the new x_i.

For each selected bitfield, s_i is approximated according to a Gaussian distribution, using a predefined Gaussian step matrix:

$$\mathbf{S} = \begin{pmatrix} \gamma_1 & \delta_1 \\ \vdots & \vdots \\ \gamma_D & \delta_D \end{pmatrix}$$

where γ_i and δ_i are the mean and standard deviation, respectively, of step s_i. For $\delta_i \leq 0.0$, **B M'** switches automatically to fixed $s_i = \gamma_i$.

Thus, **B_M'** is parameterized by n_m and **S**.

4.2 The SUB case

SUB problems can be classified according to whether the subset that must be found has a size that is *fixed* (known) in advance or is *unknown*.

4.2.1 Representation

Candidate solutions for both unknown-size SUB problems and fixed-size SUB problems can be represented using either *binary* subset encoding or *direct* subset encoding; for details on these types of encoding, the reader is referred to Chapter 2 (Section 2.2) in this thesis.

When using binary subset encoding for fixed-size SUB problems, the number of 1s in any bitstring representing a candidate subset must remain constant upon modification by an exploration operator (recombination or mutation). Bit-level exploration operators that comply with this encoding constraint can be designed in several ways, but such operators are not included in GATES, as the latter already has another set of exploration operators – based on direct subset encoding – to deal with fixed-size SUB problems (see below).

When using binary subset encoding for unknown-size SUB problems, the number of 1s in any bitstring representing a candidate subset is allowed to change upon modification by an exploration operator. Among unconstrained bit-level exploration operators of this kind are the aforementioned, heuristically simple **B MX**, **B UX** (Section 4.1.2), and **B_M** (Section 4.1.3). Thus, genetic algorithms for unknown-size SUB problems can, in fact, be developed from the NUM prototype. (Thereby, the options **B_MX'**, **B_UX'**, and **B_M'**, too, are legal, but no good reason can be given as to why these should lead to better search performance, since bitfields are meaningless in non-NUM contexts.) The bitstrings can be decoded (for evaluation purposes) by a special procedure in GATES, producing a vector of indices which specify the positions of the 1-bits.

Hereafter, SUB GAs, SUB configuration, and SUB options, are understood to address fixed-size SUB problems, using direct subset encoding – the case to which "the" SUB prototype in GATES is confined.

In brief, direct subset encoding can be accomplished as follows (Lucasius & Kateman, 1992). A candidate subset is represented as the leftmost l elements in a permutation of L ($> l$) elements; the latter comprise the source set. The rightmost $L - l$ elements represent the so-called complementary subset.

4.2.2 Recombination

The recombination operators among SUB options are: **D_SX** and **D_MX**.

Recombination operators for k-way partitioning problems in general – among which SUB problems are special in that two partitions must be found (*viz.* the subset and the complementary subset, $k = 2$) – have only recently become available (Bhuyan *et al.*, 1991; von Laszewski, 1991; Jones & Beltramo, 1991; Falkenauer, 1991a; Falkenauer, 1991b). However, for reasons given in Chapter 12 (Section 3.3.3) in this thesis, these operators feature a considerable computational overhead when

applied to fixed-size SUB problems. **D_SX** and **D_MX** elude this burden.

The principal preservational properties for SUB problems are (Chapter 2 (Section 5.3.2) in this thesis; Lucasius & Kateman, 1992): element *identity*, element *position*, element *order*, and element *adjacency*. When only identity information is important for the search task, we speak of a regular SUB problem; otherwise, the problem is called a sequenced SUB problem (Chapter 2 (Sections 2.2 and 5.4.2) in this thesis; Lucasius & Kateman, 1992).

The **D SX** operator – general-purpose subset recombination – can be used in both regular mode, i.e. identity preserving mode, and sequenced mode; the latter mode falls apart into position- and order preserving mode. For a detailed description of **D_SX**, the reader is referred to Chapter 5 (Section 4) in this thesis; an analytical-chemical application of the operator is described in (Wehrens *et al.*, 1993a,b).

D_SX, like other combinatorial recombination operators, uses sophisticated – i.e. intelligent, comparatively complex – recombination heuristics in order to comply with the relevant preservation task, as well as with the encoding constraints (see Chapter 2 (Section 5.3.1) in this thesis). As a consequence, in the process of building the child strings, the position of elements in both parent strings needs to be looked up frequently. It is therefore important that element look-up takes place fast. In GATES, this is accomplished as follows. Each string that represents a subset of l elements is purposefully stored in memory as a permutation of $L > l$ elements (i.e. including the complementary subset), as described above. In this way, the so-called inverse of the permutation can be determined. Stored in memory, this inverse permutation serves as a fast look-up table. Further details of this strategy are described in Section 4.3.2 below.

The **D_MX** operator – mix subset recombination – is restricted to regular SUB problems, and has the advantage that complementary subsets do not need to be stored in memory. For a detailed description of **D MX**, the reader is referred to Chapter 2 (Section 5.4.2) in this thesis; an analytical-chemical application of the operator is described in (Lucasius *et al.*, 1993d).

4.2.3 Mutation

The mutation operators among SUB options are: **D_RM** and **D_TM**.

The **D_RM** operator – reorder mutation – is applied to one string with length l (in elements) by conducting the following procedure n_m times. First, a segment of length l' is selected randomly on the string ($2 \leq l' \leq l$); l' is a control parameter. Then, a pair of different elements on the segment is selected randomly and the elements in the pair are swapped; hence, l' affects the extent to which a mutation is local. Incidentally, **D RM** is the equivalent of "scramble sublist mutation" described in (Davis, 1991; Syswerda, 1991).

The **D_TM** operator – trade mutation – is applied to one string with length l (in elements) by conducting the following procedure n_m times. One element, selected randomly in the string, and another element, selected randomly in the complementary string, are swapped.

4.3 The SEQ case

As yet, no SEQ applications have been developed using GATES, but this is likely to change in the near future.

4.3.1 Representation

For a discussion on permutation encoding for SEQ problems, the reader is referred to Chapter 2 (Section 2.3) in this thesis.

4.3.2 Recombination

The recombination operators among SEQ options are: **P_PMX**, **P_OX1**, **P_OX2**, **P_CX**, and **P_EX**.

In general, recombination operators for SEQ problems, like recombination operators for SUB problems, use comparatively complex recombination heuristics in order to comply with the chosen preservation task and the encoding constraints. Except for element identity, the preservational properties for SEQ problems are the same as those for SUB problems. (Element identity is an irrelevant preservational property, because all possible candidate permutations for a particular SEQ problem contain the same elements, by definition.)

Like in the SUB case, in the SEQ case, too, the nature of the recombination heuristics is such that in the process of building the child strings, the position of elements in both parent strings needs to be looked up frequently. In order to accomplish fast look-up of elements, GATES uses the *inverse* of the permutation as a look-up table. The strategy is as follows.

Consider a permutation **q**, e.g. the bottom row of integer values in:

position	0	1	2	3	4	5	6	7
element	5	4	3	1	7	6	2	0

Then, the inverse permutation, $\mathbf{q}'$, is defined by the property that q'_j is the position, i, of element j in **q**; that is, $q'_j = q'_{q_i} = i$. Thus, in the example, $\mathbf{q}'$ is the bottom row of integer values in:

index	0	1	2	3	4	5	6	7
value	7	3	6	2	1	0	5	4

(Incidentally, it is easily verified that the inverse of the inverse of **q**, is **q**.) In GATES, the following macro is used to look up an element in a permutation using the inverse permutation:

```
#define WHERE_IS(Element, InversePermut) \
        InversePermut[Element]
```

The SEQ recombination operators in GATES, summarized shortly, are compatible with the definitions in the literature, whereby **P_PMX**, **P_OX1**, and **P_EX** have the added feature that they can optionally be flagged to consider permutations as Hamiltonian cycles instead of -paths; this option is explained in Chapter 3 (Section 2.2) in this thesis. For a detailed description of **P_PMX**, **P_CX**, and **P_OX2**, the reader is referred to Chapter 2 (Section 5.5.1) in this thesis.

P_PMX – partially matched recombination – was developed by Goldberg & Lingle (1985) (Goldberg, 1989; Starkweather *et al.*, 1991; Fox & McMahon, 1991). In a child string produced by this operator, the element position, -order, and -adjacency information in the contribution from one of the parents is preserved.

P_OX1 – order-based two-point recombination – was developed by Davis (1985) (Starkweather *et al.*, 1991; Fox & McMahon, 1991). In a child string produced by this operator, the element position, -order, and -adjacency information in the contribution from one of the parents is preserved; the element order information in the contribution from the other parent is also preserved.

P_OX2 – order-based uniform recombination – was developed by Syswerda (1991) (Davis, 1991). In a child string produced by this operator, the element position, -order, and -adjacency information in the contribution from one of the parents is preserved; the element order information in the contribution from the other parent is also preserved.

P_CX – cycle recombination – was developed by Oliver *et al.* (1987) (Starkweather *et al.*, 1991; Fox & McMahon, 1991). In a child string produced by this operator, the element position, -order and -adjacency information in the contribution from one of the parent strings is preserved; the element position information in the contribution from the other parent string is also preserved.

P_EX – edge recombination – was developed by Whitley *et al.* (1989,1991) (Starkweather *et al.*, 1991; Fox & McMahon, 1991). In a child string produced by this operator, the element adjacency information contained in the overall contribution from both parent strings is optimally preserved. An enhanced version of the operator is described in (Starkweather *et al.*, 1991), which can optionally be flagged to consider permutations as Hamiltonian cycles instead of -paths; this version is implemented in GATES, as pointed out above.

4.3.3 Mutation

The only mutation operator among SEQ options is: **P_RM**, which is equivalent to **D_RM** (reorder mutation), described in Section 4.2.3 above.

5 Empirical complexity analysis: a utility for configuration

As pointed out in Part 1 (Lucasius & Kateman, 1993c), the optimization of genetic configuration, or genetic configuration for short, is usually a complex and normally time-consuming task. Several approaches already exist for some time, but none of these can be qualified as resoundingly successful (Lucasius & Kateman, 1993b).

Recently, a new approach was suggested (Manderick *et al.*, 1991), which appears to have some important advantages over the traditional approaches. This approach borrows from a methodology recently introduced (Kauffman, 1989; Wein-

berger, 1990) (see also (Ruthen, 1993)): the empirical analysis of problem complexity, based on an explorative walk in the search space using the search method of interest – or *empirical complexity analysis* for short.

It is important to realize that an empirical complexity analysis can be applied for any search method besides a genetic algorithm, e.g. simulated annealing (Aarts & Korst, 1989). The approach is universal in another sense as well: it can be applied for any target problem for which an objective function is available; this is an important advantage over the aforementioned causal complexity analysis (Frame 2), which insists on foreknowledge about correlations between the problem parameters.

An empirical complexity analysis comprises two stages:

1. explorative walk;
2. statistical analysis.

The first stage is implemented as a standard option in each of the three standard prototypes in GATES; thereby, a file is generated that contains a time-series. The second stage is implemented as a separate executable program – `autocorr` – which analyzes the time-series; usage of `autocorr` is elucidated below.

5.1 Concept

Here, the idea behind empirical complexity analyses is briefly discussed in order to show that the approach is, in principle, suitable to support genetic configuration.

Basically, the aim of an empirical complexity analysis is to estimate the so-called *apparent complexity* of a particular problem (Lucasius & Kateman, 1993a). Apparent problem complexity may be defined in several ways, e.g. as the expected number of evaluations which a particular search heuristic needs to perform in the search space in order to find the true solution. It is important to realize that, in general, the apparent problem complexity depends on both the problem and the search heuristic. The adverb "apparent" emphasizes this fact, i.e. that the problem complexity is "perceived" by the search heuristic and is not necessarily the true problem complexity. The true problem complexity is considered a pure property of the problem in that it is defined as the lowest apparent problem complexity feasible at all. A search heuristic for which this is the case is assumed to exist in the space of all possible search heuristics, but there is no general methodology available to find such a search heuristic. Hence, for many problems the true problem complexity can not be determined; that is, any quest for a search heuristic that minimizes the apparent problem complexity is bound to be based, at least in part, on trial and error and is, in principle, indefinite. In this light, the choice of genetic methodology for a particular problem can be seen as an educated guess at a subspace among all possible search heuristics, and genetic configuration as a search within this subspace, aimed at minimizing the apparent complexity of the target problem.

5.2 Complexity criteria

In view of the large search space associated with many target problems of practical interest, the above definition of apparent problem complexity it is not practicable. Better yardsticks for apparent problem complexity turn out to be statistical estimates of the ruggedness of the fitness landscape as "perceived" by the search heuristic used; intuitively, a more rugged fitness landscape is associated with a larger apparent problem complexity.

Recall from Chapter 1 (Section 3.4) in this thesis, that the fitness landscape results from the combination of the search space (i.e. the chosen problem representation) and the objective function. In general, the value returned by the objective function can be seen as an *observation* (a measurement) performed in the search space, indicating either "quality" (in maximization problems) or "error" (in minimization problems); in genetic algorithms, the former is the raw fitness, whereas the latter is cast into a raw fitness by some inverting transformation. Unlike a genetic algorithm, an empirical complexity analysis considers both kinds of observations meaningful, without any need for scaling. For this reason, we prefer to speak hereafter of an "observation landscape" – or "o-landscape" for short – rather than a "fitness landscape".

The ruggedness of an o-landscape can be estimated as follows. Using the search heuristic of interest, an explorative walk is performed on the o-landscape, whereby a time-series of observations is obtained. Subsequently, this time-series is

subjected to a statistical (auto)correlation analysis from which the aimed estimate is calculated. The reliability of the estimate is assumed to depend solely on the chosen length of the walk, i.e. it is assumed that the o-landscape is statistically isotropic for the search heuristic used. Although this assumption seems rather strong, Weinberger (1990) pointed out that an isotropic o-landscape is a reasonable assumption for many complex problems. This assumption implies that the length of a walk for which a reliable estimate of the ruggedness of an o-landscape can be obtained, is normally many orders of magnitude smaller than the average length of a search which ends in the true solution; a length in the range of [1000, 10000] observations turns out to be realistic in many cases. In other words, the assumption implies that a reliable estimate of apparent problem complexity can be obtained at comparatively low computational cost. Incidentally, it is important to realize that the lowest apparent complexity practically feasible for a particular problem, is not necessarily achieved for a search heuristic confined to performing steps to nearest neighbours in the search space.

Two types of empirical complexity analyses are considered practically important. In the first type, an *uninterrupted* explorative walk of some predefined length is performed – resulting in a time-series of observations wherein the first observation is located randomly on the o-landscape, and all other observations are located at positions obtained by applying the search heuristic to the position of the preceding observation (Figure 2). This time-series is next subjected to an autocorrelation analysis (Massart *et al.*, 1988): from the *autocorrelation function*, $\varphi(\tau)$, derived from the time-series, one derives the *autocorrelation length*, $\tau_{0.5}$ (defined by $\varphi(\tau_{0.5}) = 0.5$); a smaller $\tau_{0.5}$ indicates a more rugged – less smooth – o-landscape as "perceived" by the search heuristic used, i.e. a larger apparent problem complexity. Incidentally, the autocorrelation function may further be useful for visual inspection, as its shape, too, may contain important information (i.e. information not captured in $\tau_{0.5}$).

Abovementioned program **autocorr** can be used to perform an autocorrelation analysis, for instance by the following command:

```
autocorr  tseries report 100
```

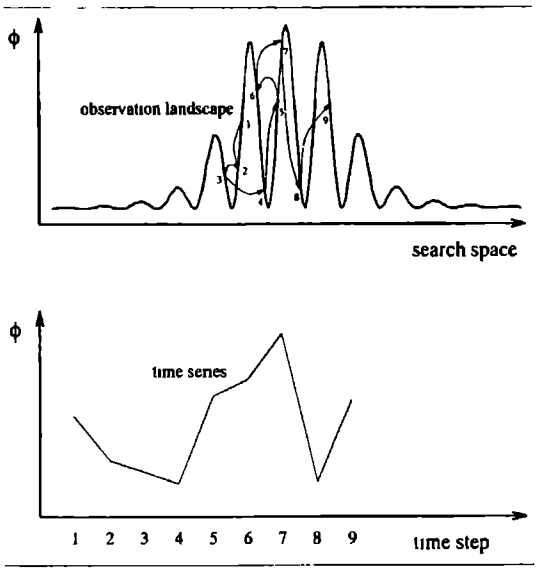

Figure 2: Top: uninterrupted explorative walk on the observation landscape for objective function ϕ. Bottom: time series of observations used for autocorrelation analysis.

The first argument (**tseries**) is the name of the input file that contains the time-series of the explorative walk; the second argument (**report**) is the name of an output file to which the value of $\tau_{0.5}$ and the values of $\varphi(\tau)$ are written; the third argument (100) is the chosen maximum for τ, which must be smaller than the length of the explorative walk.

In the second practically important type of empirical complexity analysis, a *pairwise interrupted* explorative walk of some predefined length is performed – resulting in a time-series of observations wherein the first and all other odd-indexed observations are located randomly on the o-landscape, and all even-indexed observations are located at positions obtained by applying the search heuristic to the position of the preceding (odd-indexed) observation. This time-series is next subjected to a correlation analysis: the *correlation coefficient*, r, which quantifies the correlation between the odd-indexed and even-indexed observations, is determined; a smaller r indicates a more rugged o-landscape as "perceived" by the search heuristic used, i.e. a larger apparent problem complexity.

Abovementioned program **autocorr** can also be used to perform a correlation analysis. To that end, a command similar to the one above must be

specified, only now the third argument must be 0 in order to effectuate this mode, and the output written to **report** is the value of r.

Both $\tau_{0.5}$ and r capture information about the global properties of the o-landscape (i.e. about the problem) and about the properties of the search heuristic, but with different emphasis. Intuitively, $\tau_{0.5}$ seems to capture more of the former kind of information than r does; as for the latter kind of information, the opposite seems to be true.

It is important to realize that $\tau_{0.5}$ and r, like other measures of apparent problem complexity, will generally depend on how observations are scaled. Scaling being essentially a strategy to transform an objective function into another objective function (for the same problem), $\tau_{0.5}$ and r will generally depend on the choice of the objective function, too. This implies that comparisons between apparent complexities for different target problems generally make no sense (for, different target problems have different objective functions, by definition), even when the same search heuristic is used in each case. In contrast, comparisons between apparent complexities for different search heuristics applied to the same problem, using the same objective function (including scaling), do make sense: the search heuristic that features the lowest apparent problem complexity may be considered the best choice for tackling the problem concerned. In a similar way, empirical complexity analyses can be used for configuration purposes, too, e.g. for genetic configuration, discussed next.

5.3 Genetic configuration

This section briefly describes two approaches to genetic configuration using empirical complexity analyses; the first approach is discussed merely to emphasize the advantages of the second.

In the first approach, the search heuristic that performs the explorative walk is the genetic algorithm that results from a particular genetic configuration. In the time-series completed after the walk, each observation is the fitness of the best string in the population of a particular generation; the apparent problem complexity is estimated by an (auto)correlation analysis as described above. Unfortunately, this approach features some serious drawbacks which are essentially similar to those of a traditional approach to configuration, namely: the configuration space that must be scanned is high-dimensional, often prohibitively large in practice. Still, the new approach is probably the least inefficient of the two, as it validates genetic configuration by fitnesses monitored during a genetic search that does not necessarily have to converge (assuming an isotropic o-landscape); in a traditional approach, on the other hand, genetic configuration is normally validated by the fitness of the end-solution obtained after a genetic search that has converged.

In the second approach, the search heuristic that performs the explorative walk is either a recombination operator or a mutation operator, parameterized in a particular way; technical details are discussed in the Section 6.1 below. A number of recombination operators and a mutation operators, and parameterizations thereof, are tried. In all cases, or *partial* genetic configurations, the apparent problem complexity is estimated by an (auto)correlation analysis as described above. The partial genetic configurations that feature the lowest problem complexity are, according to the findings of Manderick *et al.* (1991), still acceptable components in an overall genetic configuration. Hence, once exploration modules have been optimized separately and are kept fixed, the exploitation modules can be optimized simultaneously next. Concluding, the above approach is promising, since the genetic configuration problem can be subdivided into a number of lower-dimensional problems, leading to an exponential reduction in the overall computational burden of genetic configuration.

6 Use of the NUM prototype

This section describes the use of the NUM prototype in GATES in some detail.

6.1 General issues

A simple way to develop a NUM GA starts with writing a call to the function **NUM Prototype()** – the NUM prototype – from the function **main()** (where, in any C-program, execution begins), e.g. as follows:

```
int main(int argc, char *argv[])
{
   (void) NUM_Prototype(argc, argv);
}
```

Mode *a*: `numga <Domain> <Config> <Rseed>`

Mode *b*: `numga <Domain> <Config> <Rseed> <Explorer> <WalkLength> <Tseries>`

Frame 3: Command forms for the two modes in which NUM GAs can be run.

This code must be stored in a separate, i.e. non-GATES, software module, say the file `numga.c`. The array of pointers `argv[]` refers to the tokens on the command-line when the program, in executable form, is called from the operating system; thereby, `argv[0]` is the name of the program (the command name) and `argv[i]` is the name of the i^{th} argument ($i \geq 1$); `argc` is the number of tokens on the command-line, i.e. 1 + the number of command arguments.

Besides `main()`, the developer must define a number of domain-dependent functions that are presumed by, i.e. called by, `NUM_Prototype()`. The names of these functions are thus predefined by `NUM_Prototype()`, while their definition depends on the specific genetic application that the developer has in mind, of course. Further details are provided in Section 6.2 below.

Without loss of generality, and with gain in simplicity, it is assumed in the further discussion that the developer stores the definitions of all domain-dependent functions in a single file, `numga.c`, i.e. no other domain-dependent modules need to be taken into consideration. With all domain-dependent functions defined in `numga.c`, the aimed executable NUM GA is obtained in two steps, as follows. (Hereby we adopt, again without loss of generality, a UNIX C-compiler – `cc` – that complies with the ANSI standard.) In the first step, `numga.c` is compiled:

```
cc  -c numga.c
```

whereby an object module – the file `numga.o` – is created. In the second step, `numga.o` is linked to GATES in order to obtain the executable NUM GA:

```
cc  -o numga numga.o gates.a -lm
```

where `numga` is the desired NUM GA (its name is chosen by the developer), `gates.a` is the archive GATES, and `-lm` serves to link in the standard archive with mathematical functions as well. (An archive is understood to be a library – a file that contains a collection of object files.)

Basically, `numga` can be used in two modes, depending on the number of command-line arguments. When called with three arguments, `numga` results in "genetic algorithm" mode. Usage is then indicated by the command form specified in Frame 3a; the three arguments are explained as follows:

`<Domain>` :
The name of a user input file that contains domain information, needed for the parameterization of the objective function. This file is formatted at the discretion of the developer and is read by a compatible domain-dependent function, `_DomainHousekeep()`, elucidated in Section 6.2 below.

`<Config>` :
The name of a user input file for the domain-independent side of NUM configuration. This file has a predefined format and is read by the domain-independent function `NUM_Housekeep()`.

`<Rseed>` :
An arbitrary integer in the range [0, 900000000] that serves as the seed for the pseudo-random number generator. Results of a run can be reproduced by seeding the generator with the initial seed of a previous run, as pointed out in Section 2 above.

When called with six arguments, `numga` results in "explorative walk" mode. Usage is then indicated by the command form specified in Frame 3b. The first three arguments have the same explanation as given above; the other three arguments are explained as follows:

`<Explorer>` :
The character `R`, `r`, `M` or `m`.

When R or r is specified, the recombination operator, as selected and parameterized by <Config>, is used to perform an explorative walk. Thereby, in each pair of strings that comes up for recombination, one string is a mutated copy of the other – using the mutation operator, as selected and parameterized by <Config>; without such mutation, the recombination would not result in exploration. In each step, the observation written to the file <Tseries> is for only one (arbitrary) string in the pair; however, when a non-whitespace character, say #, is appended to R or r (leading to R# or r#), then the average observation of the string pair is written to <Tseries>.
When M or m is specified, the mutation operator, as selected and parameterized by <Config>, is used to perform the explorative walk. Observations are written to <Tseries>.
When r or m is specified, an uninterrupted- – otherwise (R or M) a pairwise interrupted – explorative walk is performed.

<WalkLength> :
An integer that specifies the number of observations to perform during the explorative walk, i.e. the length of the resulting time-series of observations.

<Tseries> :
The name of a file to which the time-series of observations monitored during the explorative walk, are written.

Other command forms as the two specified in Frame 3 can be accomplished by not passing the arguments of main() directly to NUM_Prototype(); this more advanced issue is beyond the scope of this paper, however.

6.2 Functional organization

NUM_Prototype() is stored in the object file num.o archived in GATES. In the corresponding source file (num.c), NUM_Prototype() is defined as specified in Frame 4. The conventions concerning the names of variables and functions referenced by NUM_Prototype() are as follows.

Variables of which the name starts with "_" (e.g. _Generation) are so-called standard variables; these are by definition global, hence can be accessed anywhere by the developer. There are 42 standard variables, many of which do not need to be accessed directly by the developer, as that is already done automatically in run-time. For instance, many standard variables are assigned constant (or at least initial) values by the function NUM_Housekeep(); these values are retrieved or derived from the NUM configuration file, read by this function; for example, _StringCount (the population size) is set to 100, say. Some standard variables assigned by NUM_Housekeep() are pointers, e.g. the variable _ParamValues; their value is the starting address of a block of memory that is dynamically allocated by this function. Besides write-access, read-access, too, is performed automatically on many variables in run-time; in this respect, the variables _StringCount and _ParamValues are, again, good examples.

The declaration of the standard variables can be accomplished by incorporating the line:

```
#include num_var.h
```

at the top of one of the domain-dependent modules – numga.c in the present discussion. (The header file num_var.h comes separately with GATES.)

The functions ScaleFitnesses() and Select() are domain-independent functions, i.e. they are also used by the SUB- and SEQ prototypes. ScaleFitnesses() (in scale.o in GATES) scales fitnesses in a sequence of simple steps, described Section 3.1 above. Select() (in select.o in GATES) integrates three tasks described in Sections 3.2–3.4 above: (1) full reproduction; (2) random pairing; (3) full replacement. Implementationally, this deviates somewhat from the generic flowchart referenced in Section 2 above, as replacement here takes place before recombination and mutation are applied. However, recombination and mutation in NUM_Prototype() are – not visibly for the developer – performed on the population instead of the temporary population, which has the same size (full replacement); this makes NUM_Prototype() functionally equivalent to the generic flowchart.

The functions of which the name starts with "NUM_" are domain-independent within the NUM

```
#define IF_BAD_COMMAND_LINE_ARGUMENTS \
        if ((Argc != 4 &&  Argc != 7) || \
            (Argc == 7 &&  *Argv[4] != 'r' && *Argv[4] != 'R' && \
                           *Argv[4] != 'm' && *Argv[4] != 'M'))

#define EVOLVE \
        if (Argc == 4) \
           while (_Generation < _MaxGeneration
                  && _Proceed())


void NUM_Prototype(int Argc, char *Argv[])
{
   IF_BAD_COMMAND_LINE_ARGUMENTS
      NUM_PrintUsageAndExit(Argv[0]);

  _Herald();
  _DomainHousekeep(Argv[1]);
   NUM_Housekeep(Argc, Argv);
  _Init();

   EVOLVE
   {
     _EvalStrings();
      ScaleFitnesses();
     _ShowResults();
      Select();
      NUM_RecombineStrings();
      NUM_MutateStrings();
     _Next();
   }
   else
      NUM_ExplorativeWalk(Argv);

   NUM_UnHousekeep();
  _DomainUnHousekeep();
  _Exit(0);
}
```

Frame 4: Definition of the NUM prototype.

application scope. Like `NUM_Prototype()`, these functions are in abovementioned `num.o`.

The functions of which the name starts with "_" are domain-dependent. As pointed out in the previous section, these functions must be defined by the developer. Some general guidelines to that end are the following:

`_Herald()` :
Serves to print an initial message to screen, e.g. the name and a brief description of the genetic application concerned, the names of its developers, copyrights, etc.

`_DomainHousekeep()` :
Serves to assign values to variables declared by the developer, i.e. domain-dependent variables. These possibly include pointers to dynamically allocated memory that accommodates domain-dependent data, e.g. an experimental spectrum. The information required for these purposes is normally read from a special user input file, as was indicated in Section 6.1 above.
In addition, the function may serve to write the NUM configuration file that `NUM_Housekeep()`, called next (Frame 4), expects as user input; the format of this file must, of course, comply with the predefined format. In this roundabout way, the developer gains control over the format of the required domain-independent user input as well. (The freedom to control all user input is considered important by many developers.)

`_Init()` :
Serves to do certain things, before the evolution cycle is started, that can not be anticipated on by GATES. For instance, `Generation` is set to an initial value 3, for whatever reason the developer may have; this overrides the default initial value 0 set by `NUM_Prototype()`. Or, the population is initiated according to some heuristic the developer may find useful to apply; or, the population is initiated according to a previously saved population read from file; in both cases, the default random initiation performed by `NUM_Prototype()` is overridden. Or, a screen graphics mode is entered if it is the intention of the developer that the application displays graphics.

`_Proceed()` :
Serves to apply a continuation criterion (the opposite of a termination criterion), e.g. one based on some convergence criterion (e.g. population variance). If it fails, the function returns a logical false value and the evolution cycle is exited. If it succeeds, the function returns a logical true value and the evolution cycle is continued, provided that the maximum-generation criterion (`_Generation < _MaxGeneration`) is logically true at that point.

`_EvalStrings()` :
Serves to iteratively evaluate all `_StringCount` (N) strings in the population, as follows (Frame 5). The function `NUM DecodeString()` – in `num.o` in GATES – decodes the i^{th} bitstring in the population to obtain D real values (for the D problem parameters) written to a vector block of memory pointed to by `_ParamValues`. These values are passed to the `ObjectiveFunction()` in order to calculate both the observation and raw fitness for the i^{th} bitstring; this function must be defined by the developer. (Observations and raw fitnesses are identical in maximization problems; in minimization problems they relate inversely to each other.) After the `for`-loop is completed, the vector blocks of memory pointed to by `_Observs` and raw `_Fitnesses`, respectively, have become updated for the present generation.

`_ShowResults()` :
Serves to periodically write and/or append information on the progress of the optimization procedure to screen and/or log files, e.g. the decoded form of the currently best string and its fitness. Also, the population may be saved intermittently (e.g. every 100 generations) to a special file as a provision against casual system failures; in order to resume a run, then, the function `Init()` must be defined such that it reads the desired population from file in a compatible way.

`_Next()` :
Serves to arrange certain things before the next evolution cycle is executed. For instance, `Generation` is incremented by 1. Of course, if `_Generation` would not be iteratively updated, the above-mentioned maximum-generation criterion becomes meaningless. The updating is also important for `_ShowResults()` if `_Generation` is included in the results periodically written to screen and/or file in run-time.

`_DomainUnHousekeep()` :
Serves to free (deallocate) any memory that was previously allocated dynamically by `_DomainHousekeep()`.

`_Exit()` :
Serves to do certain things before overall execution is ended, e.g. to exit a screen graphics mode, or to write the best-ever encountered solution to a file in a user-friendly form. The integer passed as an argument to the function is the code which `NUM Prototype()` returns to the operating system; this is useful for any reason the developer may have.

When in any of the above domain-dependent functions no action is supposed to be undertaken, the

```
void _EvalStrings(void)
{
   int i;

   for (i = 0; i < _StringCount; i++)
   {
       NUM_DecodeString(i, _ParamValues);

       ObjectiveFunction(_ParamValues,
                         _Observs   + i,
                         _Fitnesses + i);
   }
}
```

Frame 5: Function for the evaluation of all strings in the population.

developer should simply specify a void function body ({ }).

6.3 Configuration

The selection of genetic operators and their parameterization is controlled by the NUM configuration file, read by `NUM_Housekeep()`. Although this file (like the SUB- and SEQ- configuration files) has a predefined format, formatting restrictions are minimal. These are that the entries should each be preceded by a colon character (:) and specified in a predefined order; thereby, compound entries – i.e. entries which are designated to take the form of a list of whitespace-separated entries – are also considered as entries. Any colon-free text apart from the formal colon-entry pairs, is allowed (e.g. for commenting purposes), as it is simply ignored.

Next, we discuss a transcript of an exemplary NUM configuration file for a 10-dimensional problem; the successive entries are, for the sake of clarity, indicated by serial indices – 00 to 24 – right before the colon (thus not violating the formatting restrictions):

```
PROBLEM PARAMETERS

   NUMBER OF      (= $)
     [int, > 0]                          00:  10

   NAMES
     [$ tokens]                          01:  x0 x1 x2 x3 x4 x5 x6 x7 x8 x9

   FLOORS
     [$ reals]                           02:  0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

   CEILINGS
     [$ reals]                           03:  1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

   CENTERS
     [$ reals]                           04:  .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

   DEVIATIONS
     [$ reals]                           05:  .5 .5 .5 .5 .5 .5 .5 .5 .5 .5

   FORMAT
     [$ ints, >or= 1, <or=32]            06:  8  8  8  8  8  8  8  8  8  8

   PLACEMENT
     [$ unique ints,
      minimum 0, maximum $-1]            07:  8  1  6  3  4  5  2  7  0  9

MAXIMUM GENERATION                       08:  200
  [int, > 0]

POPULATION SIZE
  [int, > 1]                             09:  100

FITNESS

   OFFSET
     [real, >or= 0.0]                    10:  0.0

   SCALE FACTOR
     [real, > 1.0]                       11:  3.0

   SCALE MODE
     [0 = sigmoid
      1 = static linear
      2 = dynamic linear]                12:  0
```

```
SELECT

   ELITIST FRACTION
     [real, >or= 0.0, <or= 1.0]            13:  0.05

   TRESHOLD FRACTION
     [real, > 0.0, <or= 1.0 --
      ignored for select-mode 0]           14:  0.1

   SELECT MODE
     [0 = roulette    1 = treshold]        15:  0

RECOMBINE

   MODE
     [0,1 = multipoint   2,3 = uniform
      0,2 = bits         1,3 = bitfields] 16:  2

   PROBABILITY
     [real, >or= 0.0, <or= 1.0]            17:  0.9

   NUMBER OF INTER-STRING

          BREAKPOINTS
            [int, > 0 --
             ignored for modes 2,3]        18:  2

          SWAPS
            [int, > 0 --
             ignored for modes 0,1]        19:  30

MUTATE

   MODE
     [0,1 = fixed     2,3 = distributed
      0,2 = bits      1,3 = bitfields]     20:  0

   INNER-STRING

          BIT(FIELD) CHANGE PROBABILITY
            [real, >or= 0.0, <or= 1.0 --
             ignored for modes 0,1]        21:  0.01

          NUMBER OF BIT(FIELD) CHANGES
            [int, >or= 0 --
             ignored for modes 2,3]        22:  5

   STEPS
     [$ entries of the format
      <mean><colon><deviation>,
      <mean> and <deviation> are
      base-10 ints, >or= 0 ---
      ignored for modes 0,2]               23:  10:5 10:5 10:5 10:5 10:5
                                                10:5 10:5 10:5 10:5 10:5

DECODE MODE
  [0 = regular binary    1 = Gray binary] 24:  1
```

The 25 entries are briefly explained as follows:

00: D – the problem dimensionality (the number of problem parameters), denoted as **$** in the remainder of the configuration file at hand.

01: A compound entry formed by D symbolic names, chosen at the discretion of the user of the application, for the respective problem parameters. Typically, these names serve to be written to screen and/or file by the function `_ShowResults()`. Note that the names that have been chosen in the configuration file at hand, closely resemble $X_0, \cdots, X_{D-1}$ – the formal names used for the problem parameters in the further discussion. (Hereafter, vector indices are assumed to start counting at 0.)

02: A compound entry formed by D real values that represent floors for $X_0, \cdots, X_{D-1}$, respectively. That is, the decoded value for X_i is replaced by the i^{th} floor if the latter would be larger. This gives the user a simple tool to prevent mathematical exceptions.

03: A compound entry formed by D real values that represent ceilings for $X_0, \cdots, X_{D-1}$, respectively. That is, the decoded value for X_i is replaced by the i^{th} ceiling if the latter would be smaller. This gives the user a simple tool to prevent mathematical exceptions.

04: A compound entry formed by D real values that represent centers of the real value ranges for $X_0, \cdots, X_{D-1}$, respectively. These centers – $c_0, \cdots, c_{D-1}$ – form the so-called working point, or the center of the D-dimensional search volume.

05: A compound entry formed by D real values that represent deviations – $d_0, \cdots, d_{D-1}$ – relative to the working point in order to define the value ranges for $X_0, \cdots, X_{D-1}$, respectively. That is, the value range for X_i is $[c_i - d_i, c_i + d_i)$.

06: A compound entry formed by D integer values that comprise the string format $\boldsymbol{\ell} = (\ell_0, \cdots, \ell_{D-1})$, such that ℓ_i applies to X_i and lies in the range $[1, 32]$. The digitization resolution for X_i is $\rho_i = 2^{\ell_i}$; in the configuration file at hand, $\ell_i = 8$ (homogeneous string format).

07: A compound entry formed by D integer values that comprise the placement permutation $\mathbf{q} = (q_0, \cdots, q_{D-1})$. That is, the value for problem parameter X_i is encoded as the q_i^{th} bitfield in a bitstring. Basically, $\mathbf{q}$ is a handle for the user to implement the building block principle in **B_MX** and **B_MX'**; $\mathbf{q}$ does not affect the performance of **B_UX** and **B_UX'**, as these are positionally unbiased.

08: The generation after which the evolution cycle is supposed to exit under all circumstances.

09: N – the number of strings that comprise the population.

10: ε – the fitness offset used in translation (the first step in fitness scaling).

11: s – the sensitivity if linear fitness scaling applies; the segregation degree if sigmoid fitness scaling applies.

12: An integer that indicates the type of fitness scaling step to be performed after translation has been performed.

13: N_1'/N – the number of elitist selections / the population size; $N_1' = 0$ results in skipping elitist selection altogether.

14: N_{best}/N – the number of best performing strings in the population from which to randomly select / the population size; taken into account only if rank-based threshold selection applies.

15: An integer that indicates the type of selection to be performed after elitist selection has been performed.

16: An integer that indicates the type of recombination operator to use (either **B_MX**, or **B_MX'**, or **B_UX**, or **B_UX'**).

17: p_r – the recombination probability.

18: n_b – the number of breakpoints in **B_MX** or **B_MX'**; taken into account only if one of these recombination operators apply.

19: n_s – the number of elementary swaps in **B_UX** or **B UX'**; taken into account only if one of these recombination operators apply.

20: An integer that indicates the type of mutation operator to use (either **B_M** parameterized by n_m, or **B_M** parameterized by p_m, or **B_M'** parameterized by n_m and **S**, or **B_M'** parameterized by p_m and **S**).

21: p_m – the mutation probability from which a binomially distributed n_m is derived.

22: n_m – the number of bits in **B_M** or bitfields in **B_M'** (whichever applies) that should undergo mutation.

23: A compound entry formed by D pairs of colon-separated integers; each pair corresponds with a row in **S** – the Gaussian step matrix for **B_M'**; taken into account only if this mutation operator applies.

24: An integer that indicates the type of binary decoding of bitstrings to be performed.

7 Conclusions and outlook

The toolbox GATES has been described in some detail, with emphasis placed on the NUM prototype. GATES was our developmental base for genetic applications during the past few years; applications are spread across various fields of computational chemistry, and have led to promising results, in general. In view of recent developments towards professional genetic development software for public use, we feel that the purchase of a commercial genetic development package becomes increasingly attractive. Against this background, our paper may be of help in validating forthcoming professional genetic development software, in general.

Depending on the response on this paper, it is intended to make publicly available the NUM prototype in GATES and the separate program `autocorr`, as described in this paper, according to the terms of an agreement; the authors should be contacted for more details.

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007. We wish to thank Dr. L. Davis and Drs. A.P. de Weijer for their critical comments and useful suggestions on the draft version of this paper.

References

[1] Bhuyan, J.N., Raghavan, V.V., and Elayavalli, V.K. Genetic algorithms for clustering with an ordered representation. In Belew, R.K. and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 408–415, San Mateo, CA, 1991. Morgan Kaufmann.

[2] Caruana, R.A. and Schaffer, J.D. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In Laird, J.E., editor, *Proceedings of the Fifth International Conference on Machine Learning*, page 153, Los Altos, CA, 1988. Morgan Kaufmann.

[3] Davis, L. Applying adaptive algorithms to epistatic domains. In *Ninth International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[4] Davis, L., editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY, 1991. Part I: A Genetic Algorithms Tutorial, p. 1–101.

[5] De Jong, K.A. An analysis of the Behavior of a Class of Genetic Adaptive Systems. Technical Report 185, University of Michigan, Ann Arbor, MI, August 1975. Ph.D. Thesis.

[6] Eshelman, L.J., Caruana, R.A., and Schaffer, J.D. Biases in the crossover landscape. In Schaffer, J.D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, San Mateo, CA, 1989. Morgan Kaufmann.

[7] Falkenauer, E. A genetic algorithm for grouping. In Gutiérrez, R. and Valderrama, M.J., editors, *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis*, pages 198–206, Singapore, 1991a. World Scientific.

[8] Falkenauer, E. A genetic algorithm for conceptual clustering. Technical Report Report FMS39, CRIF Industrial Automation, Brussels, December 1991b.

[9] Fox, B.R. and McMahon, M.B. Genetic operators for sequencing problems. In Rawlins, G.J.E., editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, pages 284–300, San Mateo, CA, 1991. Morgan Kaufmann.

[10] Goldberg, D E and Lingle, R Alleles, loci, and the traveling salesman problem In Grefenstette, J J , editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, 1985 Lawrence Erlbaum Associates

[11] Goldberg, D E *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, Reading, MA, 1989

[12] Holland, J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[13] James, F A review of pseudorandom number generators *Computer Physics Communications*, 60 329–344, 1990

[14] Jones, D R and Beltramo, M A Solving partitioning problems with genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 442–449, San Mateo, CA, 1991 Morgan Kaufmann

[15] Kauffman, S Adaptation on rugged fitness landscapes In Stein, L , editor, *Lectures in the Sciences of Complexity, SFI Studies in the Sciences of Complexity*, page 527, Redwood City, CA, 1989 Addison-Wesley

[16] Knuth, D E *The Art of Computer Programming*, volume 3 Sorting and searching Addison-Wesley, Reading, MA, 1973

[17] Laszewski von, G Intelligent structural operators for the k-way graph partitioning problem In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52, San Mateo, CA, 1991 Morgan Kaufmann

[18] Lucasius, C B and Kateman, G Towards solving subset selection problems with the aid of the genetic algorithm In Manner, R and Manderick, B , editors, *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, pages 239–247, Amsterdam, 1992 Elsevier

[19] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 1 Concepts, properties and context *Chemometrics and Intelligent Laboratory Systems*, 19 1–33, 1993a

[20] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 2 Representation, configuration and hybridization *Chemometrics and Intelligent Laboratory Systems*, 1993b In press

[21] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 1 General perspective *Computers & Chemistry*, 1993c Accepted

[22] Lucasius, C B , Dane, A D , and Kateman, G On k-medoid clustering of large data sets with the aid of a genetic algorithm Background, feasibility and comparison *Analytica Chimica Acta*, 1993d In press

[23] Manderick, B , Weger de, M , and Spiessens, P Genetic algorithms and the structure of the fitness landscape In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150, San Mateo, CA, 1991 Morgan Kaufmann

[24] Marsaglia, G , Zaman, A , and Tsang, W W Toward a universal random number generator *Statistics and Probability Letters*, 8 35–39, 1990

[25] Massart, D L , Vandeginste, B G M , Deming, S N , Michotte, Y , and Kaufman, L *Chemometrics A textbook* Elsevier, Amsterdam, 1988

[26] Oliver, I M , Smith, D J , and Holland, J R C A study of permutation crossover operators on the traveling salesman problem In Grefenstette, J J , editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 224, Hillsdale, NJ, 1987 Lawrence Erlbaum Associates

[27] Reingold, M R , Nievergelt, J , and Deo, N *Combinatorial Algorithms Theory and Practice* Prentice-Hall, Englewood Cliffs, NJ, 1977

[28] Ruthen, R Adapting to complexity *Scientific American*, 268(1) 110–117, January 1993

[29] Spears, W M and De Jong, K A An analysis of multi-point crossover In Rawlins, G J E , editor, *Proceedings of the First Workshop on the Foundations of Genetic Algorithms*, page 301, San Mateo, CA, 1991a Morgan Kaufmann

[30] Spears, W M and De Jong, K A On the virtues of parameterized uniform crossover In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, CA, 1991b Morgan Kaufmann

[31] Starkweather, T , McDaniel, S , Mathias, K , Whitley, D , and Whitley, C A comparison of genetic sequencing operators In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69–76, San Mateo, CA, 1991 Morgan Kaufmann

[32] Syswerda, G. Uniform crossover in genetic algorithms. In Schaffer, J.D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, page 2, San Mateo, CA, 1989. Morgan Kaufmann.

[33] Syswerda, G. Schedule optimization using genetic algorithms. In Davis, D., editor, *Handbook of Genetic Algorithms*, page 350, New York, NY, 1991. Van Nostrand Reinhold. Chapter 21.

[34] Wehrens, R., Lucasius, C.B., Buydens, L., and Kateman, G. HIPS, a hybrid self-adapting expert system for NMR spectrum interpretation using genetic algorithms. *Analytica Chimica Acta*, 277:313–324, 1993a.

[35] Wehrens, R., Lucasius, C.B., Buydens, L., and Kateman, G. Sequential assignment of 2D NMR spectra of proteins using genetic algorithms. *Journal of Chemical Information and Computer Sciences*, 33(2):245–251, 1993b.

[36] Weinberger, E. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325, 1990.

[37] Whitley, D., Starkweather, T., and Fuquay, D. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In Schaffer, J.D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, San Mateo, CA, 1989. Morgan Kaufmann.

[38] Whitley, D., Starkweather, T., and Shaner, D. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In Davis, D., editor, *Handbook of Genetic Algorithms*, pages 350–372, New York, NY, 1991. Van Nostrand Reinhold. Chapter 22.

CHAPTER 5

Towards Solving Subset Selection Problems with the Aid of the Genetic Algorithm

Contents

Towards Solving Subset Selection Problems with the Aid of the Genetic Algorithm

Abstract

This paper presents a recombination operator and two mutation operators for genetic algorithms dedicated to subset selection problems. The first application of these operators, reported elsewhere in the literature, has yielded promising results.

1 Introduction

Subset selection as a field of application for genetic algorithms, has up to now received comparatively little attention. On the other hand, it turns out that subset selection problems are abundant in many applied sciences, including analytical chemistry, and that there is a need for better methods when the subset selection problem is large and complex. This coincidence motivated us to design the here presented operators for genetic algorithms dedicated to subset selection problems.

Subset selection problems, or SUB problems for short, are among combinatorial optimization problems. Another class of combinatorial optimization problems comprises sequencing problems, or SEQ problems for short. As opposed to SUB problems, SEQ problems have received considerable attention in the literature on genetic algorithms, perhaps most notably traveling salesperson problems and routing problems in engineering sciences. In order to put SUB problems in a better perspective, the next two paragraphs briefly pass in review some general issues concerning SEQ problems.

Unless strong assumptions and/or idealistic simplifications are made with regard to the problem domain, SEQ problems can not be solved in a direct (i.e. deductive, analytical) way, which is why an indirect (i.e. inductive, iterative, search-based) approach is imperative. More precisely, many SEQ problems are provably $\mathcal{NP}$-complete [2, 5]; that is, they belong to a distinctive class of problems for which it can be proven that it is most difficult to find an algorithm that solves the problem in polynomial time. In practice this means that for many algorithms the number of candidate solutions that must be sampled to find the true solution, grows exponentially with the number of problem parameters involved, i.e. with the size of strings representing candidate solutions.

In SEQ problems, representations of candidate solutions are usually built as a concatenation of *elements*: labels that uniquely represent objects in the problem domain, e.g. cities in a traveling sales-person problem. Usually, either alphabetical symbols (e.g. $a, b, c, \cdots$) or serial numbers (e.g. $1, 2, 3, \cdots$) are adopted as elements. Thus, sequencing problems may be regarded as a search for the globally optimal permutation(s) of elements – e.g. **3 2 5 1 4**, representing a 5-city tour in a traveling sales-person problem – according to some optimality criterion.

Like SEQ problems, many SUB problems, too, are known or believed to be $\mathcal{NP}$-complete. Here the problem is to extract from a given set of elements, the globally optimal subset of elements, according to some optimality criterion. In general, the size of this subset is either unknown (variable) in advance or it is assumed fixed; this paper is confined to the latter case. In addition to the selection of the optimal subset, the subset selection task might require an optimal arrangement to be found for the extracted elements. Accordingly, such SUB problems are here referred to as *sequenced* SUB problems – to distinguish them from *regular* SUB problems, where the order of extracted elements is immaterial.

This paper presents a recombination operator and two mutation operators for both sequenced- and regular SUB problems. The first application of a genetic algorithm comprising these operators was reported by Wehrens *et al.* [6, 7] – see Chapter 11 in this thesis – and has yielded encouraging results. (In the chapter at hand, references made to other parts in this thesis are assumed to imply the there used terminology and conventions, unless explicitly overridden.)

2 Information processing

In order that the search be efficient and efficacious, a genetic algorithm must be configured such that it will optimally process, or *preserve*, relevant past information upon recombination. (Mutation is normally not assigned a preservational task, as it traditionally plays a minor role in exploration.) This notion has become known as the *building block principle* [3, 4].

Information processing was formalized by Holland [4] for a simple (i.e. binary) encoding and simple genetic operators, which culminated in the so-called *schema theorem*. Later, in a quest for a more general theory and unifying concepts, several attempts have been launched to formalize information processing for more complex encodings and more sophisticated genetic operators, as well [1]. However, as may be ascribed to the complexity of genetic algorithm dynamics in general, these endeavors have met only limited success. It is for this reason that practitioners engaged in combinatorial optimization with genetic algorithms normally resort to an intuitive approach in realizing the building block principle; that is, they use educated "common sense" in designing recombination heuristics targeted at preserving the important information, or properties, for the task concerned.

Apart from the preservational task, some recombination operators also observe *encoding constraints*, depending on the chosen encoding. An example of a constrained encoding is aforementioned permutation encoding for SEQ problems: each element must occur exactly once in a string that represents a candidate solution; for further details, the reader is referred to Chapter 2 (Section 2.3) in this thesis.

Candidate subsets can be encoded in many ways. A convenient encoding is *direct subset encoding*, discussed in Section 3 below. Direct subset encoding has in common with permutation encoding, that it uses alphabetical or numerical elements in string representations of candidate solutions; again, these elements represent objects in the problem domain. Also, like permutation encoding, direct subset encoding is constrained, although in a different way: each element may not occur more than once; for further details, the reader is referred to Chapter 2 (Section 2.2) in this thesis.

Depending on the purposes of the search task, a subset recombination operator may preserve different properties in creating child strings from parent strings. We distinguish four principal preservational properties for SUB problems:

element identity;

element position;

element order;

element adjacency.

For a detailed treatise on these preservational properties, the reader is referred to Chapter 2 (Section 5.3.2) in this thesis. Here, we suffice with the following general remarks.

Any subset recombination operator must preserve element identities. Basically, this is another way of stating that the encoding constraints for direct subset encoding must be obeyed. Encoding constraints may be obeyed according to different strategies. An example is a strategy which requires that all elements common to both parent strings appear in the child strings, i.e. that the parental cross-section is preserved. Other identity preserving strategies are conceivable.

While a subset recombination operator for regular SUB problems only needs to preserve element identities, a subset recombination operator for sequenced SUB problems must additionally preserve element position, -order, and/or -adjacency. When more than one property must be preserved, the recombination operator becomes multi-objective. This may introduce the need to make inter-property compromises when optimal preservation of one property would be at the expense of another.

3 Direct subset encoding

For the discussion on direct subset encoding, the concepts *set* and *string* are defined first.

A set of elements is uniquely denoted as a sorted row of element identities between braces, e.g. {2,5,7}.

A string of elements is uniquely denoted as an unsorted row of element identities between parentheses, e.g. (7,2,5). A string uniquely defines a set. For instance, the string (7,2,5) defines the set {2,5,7}. Conversely, a set is said to span a number of strings, e.g. the set {2,5,7} spans $3! = 6$ strings, among which the string (7,2,5).

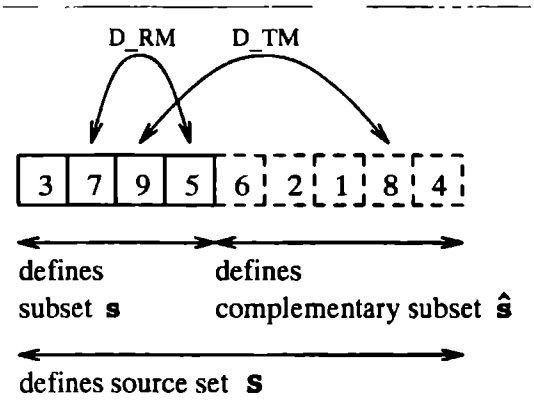

Figure 1: (a) String representation of a subset (size $l = 4$) and its complementary subset (size $L - l = 5$). (b) Reorder mutation (**D_RM**) and trade mutation (**D_TM**).

The subset selection task can now be described as follows. From a source set **S** that contains L elements, $\mathbf{S} = \{1, 2, \cdots, L\}$, select an optimal subset, **s**, of fixed size l ($l < L$), according to some optimality criterion. The subset of elements in **S** that are not in **s** is called the complementary subset, $\hat{\mathbf{s}}$.

The population consists of a fixed number, N, of candidate subsets $\mathbf{s}_i$ ($i = 1, \cdots, N$), each represented as a string of l elements selected from **S**. A random initial population can be created by randomizing N permutations, each of size L, and then to consider the first l elements in each permutation as a candidate subset. Such an encoding is referred to as an ***l/L*** ("l from L") direct subset encoding. The last $L - l$ elements in the permutations automatically define the respective complementary subsets. A schematical illustration of an encoded subset and its complement is given in Figure 1a.

4 Recombination and mutation for subsets

This section presents the *general-purpose subset recombination* operator, the *reorder mutation* operator, and the *trade mutation* operator - denoted by **D_SX**, **D_RM**, and **D_TM**, respectively. (The prepended "D" in the names serves to remind of the fact that direct, rather than binary, subset encoding is used.)

4.1 Recombination

D_SX can be used in both regular mode and sequenced mode (hence *general-purpose* subset recombination). The sequenced mode falls apart into a position- and order preserving mode; an adjacency preserving mode has not been implemented, but may be considered in the future.

In general, the creation of a child string C from parent strings P_1 and P_2 can formally be represented as:

$$P_1 \otimes P_2 \rightarrow C$$

where $\otimes$ denotes the recombination operator.

In a spirit loosely similar to **P_OX2**, discussed in Chapter 2 (Section 5.5.1) in this thesis, **D_SX** can be described as follows - using, for illustration, parent strings P_1 = **1 8 3 4 7** and P_2 = **6 2 4 9 1** in a **5/9** direct subset encoding:

1. Create a random binary template (bitstring) of a size identical to the length of a parent string, such that it can serve as a mask to select elements from parent string P_1, e.g. (with P_2 also shown):

   P_1 = 1 8 3 4 7
   0 1 0 1 1 (binary template)
   P_2 = 6 2 4 9 1

2. Copy the 1-masked elements in P_1 to the same respective positions in child string C (which thereby inherits identity, position, and order information), in our example:

   C = - 8 - 4 7

   The -'s denote undefined elements. Let n_C be the number of defined elements in C ($n_C \leq n$, in our example $n_C = 3$), and let $\bar{n}_C = n - n_C$.

3. Mark in P_2 all elements that are now defined in C, as undefined elements (in our example only 4):

   P_2 = 6 2 - 9 1

   Let n_{P_2} be the number of defined elements in P_2 ($n_{P_2} \leq n$, in our example $n_{P_2} = 4$). Note that the inequality $\bar{n}_C \leq n_{P_2}$ must apply at this stage.

4. If $\bar{n}_C = n_{P_2}$ (which does not apply to our example):

   Copy each defined element in P_2 to an undefined element in C as follows.

   (a) In **identity** preserving mode:
   Copy without any preference *per se* for position or order, e.g. randomly.

(b) In **position** preserving mode:
Wherever possible, copy a defined element in P_2 to an undefined element in C at the same position. Remaining elements are copied without any preference *per se* for order, e.g. randomly.

(c) In **order** preserving mode:
Copy the defined elements in P_2 to the undefined elements in C, retaining the order they have in P_2.

5. Otherwise ($\bar{n}_C < n_{P_2}$, which applies to our example):

Copy each element in a fraction of $\bar{n}_C$ elements among the n_{P_2} defined elements in P_2 to an undefined element in C as follows.

(a) In **identity** preserving mode:
Select the fraction randomly, then proceed as in 4(a). In our example, if the $\bar{n}_C$ (= 2) selected elements in P_2 are for instance 9 and 6, then C becomes:

C = 9 8 6 4 7

(b) In **position** preserving mode:
Select the fraction semi-randomly, i.e. a number of elements in the fraction is selected deterministically in such amount as to maximize preservation of position in the next step, which is to proceed as in 4(b). In our example this means that in P_2 6 must be selected and the other $\bar{n}_C - 1$ (= 1) elements randomly, for instance 2 in our example. C then becomes:

C = 6 8 2 4 7

(c) In **order** preserving mode:
Select the fraction randomly, then proceed as in 4(c). In our example, if the $\bar{n}_C$ (= 2) selected elements in P_2 are for instance 2 and 9, then C becomes:

C = 2 8 9 4 7

In order to obtain the second child string, the procedure should be repeated with the roles of the parent strings interchanged.

A modification of the identity preserving mode is to preserve the parental cross-section, as pointed out above. This constrains the creation of the binary template in Step 1 of the description of **D SX**. For instance, in our example, since the elements 1 and 4 comprise the parental cross-section, the binary template for P_1 must be of the form `1 ? ? 1 ?`, where the "wildcard" symbol ? denotes unconstrained bit values that can be chosen randomly. Similarly, a modification of the position- and order preserving modes is to preserve either one or two substrings of one of the parent strings in a fashion similar to recombination operators **P_OX1** and **P_PMX**, both presented in Chapter 2 (Section 5.5.1) in this thesis. Again, this constrains the creation of the binary template in Step 1 of the description of **D_SX**, namely: the binary template must be a random multipoint binary template of type $T_M(1)$ or $T_M(2)$; for details on random binary templates as a tool in the description of recombination operators, the reader is referred to Chapter 2 (Section 5.1.3) in this thesis. Examples are: `0 0 1 1 1` (two uniform segments: `0 0` and `1 1 1`), `1 1 0 0 1` (three uniform segments: `1 1`, `0 0` and `1`), `0 1 1 0 0` (three uniform segments: `0`, `1 1` and `0 0`), etc.

4.2 Mutation (Figure 1b)

D_RM repeats, a predefined number of times, a procedure that swaps two randomly selected elements within a string. To that end, any available permutational mutation can be used, too, e.g. Davis' "scramble sublist mutation" [1]. Obviously, **D RM** only makes sense for sequenced SUB problems.

D TM is useful in both regular and sequenced SUB problems. It repeats, a predefined number of times, a procedure that "trades" (i.e. exchanges in an import-export fashion) a randomly selected element in a candidate string for a randomly selected element in the complementary subset related to that string. **D TM** can be used alongside **D_RM** in sequenced subset selection problems.

For some further details on **D_RM** and **D TM**, the reader is referred to Chapter 2 (Section 4.2.3) in this thesis.

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007. We wish to thank A.D. Dane and W.J. Melssen

for stimulating discussions, critical comments, and useful suggestions

References

[1] Davis, L , editor *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York, NY, 1991

[2] Garey, M R and Johnson, D S *Computers and Intractability A Guide to the Theory of NP-Completeness* W H Freeman and Company, San Francisco, CA, 1979

[3] Goldberg, D E *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, Reading, MA, 1989

[4] Holland, J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[5] Maffioli, F The complexity of combinatorial optimization problems and the challenge of heuristics In Christofides, N , Mingozzi, A , Toth, P , and Sandi, C , editors, *Combinatorial Optimization*, page Chapter 5, New York, NY, 1979 Wiley

[6] Wehrens, R , Lucasius, C B , Buydens, L , and Kateman, G HIPS, a hybrid self-adapting expert system for NMR spectrum interpretation using genetic algorithms *Analytica Chimica Acta*, 277 313–324, 1993

[7] Wehrens, R , Lucasius, C B , Buydens, L , and Kateman, G Sequential assignment of 2D NMR spectra of proteins using genetic algorithms *Journal of Chemical Information and Computer Sciences*, 33(2) 245–251, 1993

PART III

APPLICATION

CHAPTER 6

Conformational Analysis of DNA Using Genetic Algorithms

Contents

Conformational Analysis of DNA Using Genetic Algorithms

Abstract

Among techniques for conformational analysis of DNA molecules, restrained molecular dynamics and distance geometry have, up to now, met widespread application. However, as both techniques are essentially based on a local search heuristic and feature strong domain assumptions (e.g. originating from the expert biochemist's intuition) for pragmatic reasons, their performance is poor when complex conformational problems are concerned. In order to overcome these shortcomings, it is paramount to relax the domain assumptions, e.g. to allow for non-linearities, and to employ a more global, yet efficient search heuristic. A novel, promising search technique that fits this profile, is embodied in genetic algorithms. This paper discusses a genetic algorithm, called DENISE, dedicated to conformational analysis of aqueous DNA, using atomic distance constraints and/or 2D-NOE spectra as experimental input. For the purposes of this pilot study, wherein only proof of principle needs to be assessed, simulated experimental data suffice. Results indicate that DENISE samples the search space in an efficient, robust, and productive way, as it delivers, within reasonable time, sensible conformations that satisfy all constraints.

1 Introduction

The hereditary material of all living systems, deoxyribonucleic acid (DNA), comprises a family of biological macromolecules of great importance. DNA is built up from the polymerization (chemical chaining) of nucleotides, which accounts for its variable size and different folding modes. Nucleotides are molecules of which four species exist [11]: thymidine (T), guanosine (G), cytidine (C) and adenosine (A). Three functional groups can be distinguished in each nucleotide: *backbone*, *furanose* (sugar) ring, and *base* (Figure 1). The base is a molecular ring system of which four types exist; it uniquely determines the identity of the nucleotide that it is part of.

One type of natural DNA complexes that currently enjoys widespread attention in the field of biomolecular conformational research, belongs to the class of DNA *hairpins* [1, 2, 6, 12]. These complexes are formed in partially self-complementary sequences of nucleotides – thereby forming a distinct *loop* and *stem* section (e.g. Figure 2) – and are the primary subject of our investigations.

General consensus exists among biochemists that the three-dimensional structure of DNA complexes is a key factor in their observed biological functionality (activity). It is therefore not surprising that many computational techniques have been developed for the acquisition and interpretation of structural information at the atomic level. These techniques often depend on experimental data from nuclear magnetic resonance (NMR) spectrometry. The reason for this is that, in general, biological macromolecules are abundant in atoms that have a magnetic spin moment: predominantly hydrogen atoms (H). NMR spectra contain reliable structural information, but the extraction of this information turns out to be difficult in practice [2, 12].

A suitable NMR technique for conformational analysis of biological macromolecules is two-dimensional (2D) Nuclear Overhauser Enhancement (NOE) NMR spectrometry. The integration of a 2D-NOE spectrum leads to a so-called NOE intensity matrix, or NOE table for short, which lists the "through-space" magnetic interactions between H atoms. From the theory behind NOE spectrometry, a mathematical model can be derived – known as the Bloch-Redfield relaxation model [12, 9] – which describes a relation between the relative positions of H atoms in a molecular conformation and the NOE table pertaining to it. Thus, the information contained in NOE spectra basically "pins down" the relative spatial positions of H atoms in the molecule, leading to geometrical constraints that define a subspace in the conformational space wherein the true conformation is supposed to lie.

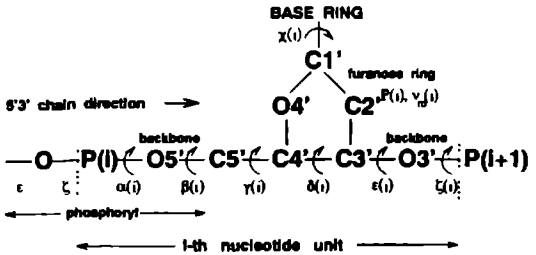

Figure 1: Schematic drawing of one (i^{th}) nucleotide unit, emphasizing backbone and furanose (sugar) ring. Three-dimensional impression is suppressed: in reality the furanose ring is puckered and bonding angles in the backbone are approximately 110°. Atoms are indexed according to international conventions. Hydrogen atoms are not drawn in order to emphasize the principal conformational parameters, which are: the torsion angles α, β, γ, ϵ and ζ in the backbone, the torsion angle χ along the bond connecting furanose ring and base ring system, and the furanose ring parameters ν_m (pucker amplitude) and P (pseudo-rotational phase angle).

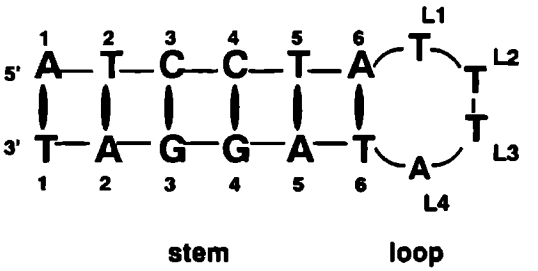

Figure 2: Schematical drawing of the hairpin d(ATCCTA-TTTA-TAGGAT), emphasizing stem and loop section. By convention, the four nucleotides in the loop section are also denoted as L_i ($i = 1, 2, 3, 4$).

2 Interpretation of NOE spectra

The Bloch-Redfield model is reputedly reliable, but also complex. More precisely, it is mathematically transparent in one direction only: from molecular conformation to NOE table, and not *vice versa*. Therefore, direct (i.e. deductive, analytical) interpretation of NOE spectra is not feasible. Instead, indirect (i.e. inductive, iterative, search-based) techniques are needed that sample the conformational space.

Sampling of the conformational space amounts to iteratively trying candidate conformations: each candidate conformation created, is evaluated by an objective function, based on e.g. the Bloch-Redfield model, in order to obtain an assessment of its utility. Using this utility, direction can be given to the search. Up to now, *restrained molecular dynamics* and *distance geometry* are the methods that are most widely applied for this purpose. Exhaustive search – or enumerative search, or grid search – is also used; this approach guarantees that the globally optimal conformation will be found, but it is computationally very expensive and therefore limited to comparatively small molecules, in practice.

In restrained molecular dynamics, a force-field is assumed between all atoms in the molecule. Starting from an educated guess at an acceptable structure (depending on the expert biochemist's heuristic knowledge), the conformational space is searched for a conformation with minimal energy; usually, a (linear) local search heuristic is used: steepest descent, or gradient, search. The conformation obtained after convergence, is subjected to the Bloch-Redfield model to calculate a NOE table. This theoretical NOE table is then compared to the experimental NOE table in order to judge the validity of the conformation (hence *restrained* molecular dynamics). If the match turns out to be poor, the entire procedure is repeated with a different starting conformation. In brief, the need to choose an "acceptable" force field, the dependence on a guessed starting structure, and the use of a local search heuristic, are believed to be the main reasons behind the poor sampling properties of restrained molecular dynamics when complex DNA molecules, e.g. DNA hairpins, are concerned.

In *distance geometry*, the search is guided by applying geometrical rules (constraints) and a starting structure is not stricktly needed. In this case, too, the sampling scheme is essentially local and the validity of the DNA conformation found is *a posteriori* verified by checking to what extent the given experimental constraints are satisfied. Again, when increasingly complex DNA molecules are involved, difficulties are seen to arise, i.e.

strongly unwinded structures are commonly obtained [10].

In order to perform successful conformational analysis of complex DNA molecules, domain assumptions need to be relaxed and a more global, yet efficient search heuristic is required. An alternative search technique that fits this profile is embodied in *genetic algorithms* [4, 5]. This idea has lead to the development of a genetic algorithm program called DENISE: DNA Evolutionary NOE Interpretation system for Structure Elucidation.

3 The genetic algorithm DENISE

Genetic algorithms comprise a large family of search techniques based loosely upon the principles of natural evolution according to Darwin; for a detailed general treatise, the reader is referred to Chapters 1 and 2 in this thesis. (In the chapter at hand, references made to other parts in this thesis are assumed to imply the there used terminology and conventions, unless explicitly overridden.)

As can be inferred from the literature, there is mounting empirical evidence that genetic algorithms perform well in tackling high-dimensional optimization problems characterized by many (sub)optimal candidate solutions, or multi-modalities. Conformational analysis of DNA is a similarly ill-conditioned (i.e. complex, large-scale) problem.

3.1 General

The conformational space taken into account by DENISE, is spanned by torsion angles along single chemical bonds in the DNA molecule (Figure 1). That the conformational space of even small DNA molecules is astronomical, may be appreciated by the following approximation. Typically, the value of each torsion angle needs to be known with a precision of 0.1% of its confined range, i.e. 10^3 levels should be considered. A DNA sequence of, say, 2 nucleotides, comprises about 16 important conformational parameters. Hence, for this 16-dimensional conformational problem, the conformational space is a collection of 10^{48} states. This size is beyond exhaustive search, in practice. For local search, on the other hand, the computation time is usually within reasonable time limits, but in view of the large number of unknown multi-modalities that may be expected in high-dimensional conformational problems, it is likely that a sub-optimal conformation will be found.

In DENISE, evaluation falls apart into two basic strategies, each of which takes different experimental input. The first evaluation strategy uses the Bloch-Redfield model to calculate NOE tables from candidate conformations – as in restrained molecular dynamics; these spectra are compared with experimental constraints in the form of a NOE table to assess the utility of the candidate conformation concerned. The second evaluation strategy uses another set of experimental constraints, namely atomic distance constraints (often estimated roughly from a NOE table) – as in distance geometry; the extent to which the distance constraints are satisfied, determines the utility of the candidate conformation concerned.

In contrast to restrained molecular dynamics and distance geometry, however, DENISE uses these experimental constraints *on-line*, i.e. not afterwards, and terminates when a predefined fraction (e.g. 0.95) of all constraints are satisfied within experimental error. Concluding, DENISE distinguishes itself from the traditional strategies in that it is highly automated, thus liberating the expert biochemist from the burden of interacting intensively with the optimization procedure.

3.2 Representation

DENISE starts from a *known* DNA sequence of nucleotides. A conformation is uniquely defined when, given the atomic connection scheme of the molecule, each bonding distance (length of a bond), bonding angle (angle in a bond pair, i.e. between two adjacent bonds), and torsion angle (angle between the planes defined by two overlapping bond pairs, e.g. γ between the planes $O_{5'}$–$C_{5'}$–$C_{4'}$ and $C_{5'}$–$C_{4'}$–$C_{3'}$ in Figure 1) in the molecule is specified. However, bonding distances and bonding angles are already known fairly accurately (from X-ray diffraction data) and may be considered reasonably constant. Thus, the torsion angles account by far for most of the molecule's flexibility and, accordingly, only the latter are regarded to define trial conformations.

The five torsion angles in the furanose ring section of each nucleotide are sterically constrained, i.e. these *endocyclic* torsion angles can be reduced to two conformational parameters: the furanose

ring parameters ν_m (pucker amplitude) and P (pseudo-rotational phase angle). The endocyclic torsion angles are given by:

$$\nu_j = \nu_m \cos\left[P + \frac{4\pi(j-2)}{5}\right]$$

where $j = 0, 1, 2, 3, 4$, for ν_j along chemical bonds $O_{4'}$–$C_{1'}$, $C_{1'}$–$C_{2'}$, $C_{2'}$–$C_{3'}$, $C_{3'}$–$C_{4'}$ and $C_{4'}$–$O_{4'}$, respectively (Figure 1).

The conformational behavior of DNA can be summarized as follows. In order of decreasing internal rotational degrees of freedom, the functional groups in DNA should be arranged as: backbone, furanose ring, and the flat base ring system. There are 8 principal conformational parameters per nucleotide unit – χ, ν_m, P, α, β, γ, ϵ and ζ (Figure 1) – which are all constrained to a finite range. The torsion angles α, β, γ, ϵ, ζ and χ, can, in principle, take all values in the range $[0°, 360°)$; δ is part of both the backbone and furanose ring (where it is called ν_3) and is therefore redundant. Ranges for P and ν_m can be derived by means of COSY (correlation NMR spectrometry), which is beyond the scope of this paper, however.

A string of values for the conformational parameters represents a conformation. These values are encoded in a binary form, in conformity with the approach encountered in the mainstream literature on genetic algorithms; the (dis)advantages of binary encoding are explained in Chapters 1 and 2 in this thesis. The binary encoded values are decoded to discrete real values in a confined range, e.g. $[0°, 360°)$ for the torsion angles; a range larger than the latter is meaningless due to periodicity. More precisely, if L is the assumed real lowbound for a particular conformational parameter, H is the assumed real highbound for the conformational parameter, B is the number of bits in the bitfield (i.e. segment within the bitstring) that encodes the value of the conformational parameter concerned, E is the encoded (i.e. integer) value of the bitfield, then the decoded (i.e. discrete real) value, D, of the conformational parameter is given by the relation:

$$D = L + (H - L)\frac{E}{2^B}$$

The bitfield size B dictates the resolution (precision) that can be attained in determining the decoded parameter value and should therefore be set as a control parameter. Our choice is $B = 10$ (hence 1024 levels), in agreement with empirically known limitations to the precision of torsion angle values: the best precision that can be attained is approximately 0.5°.

Instead of treating a particular bitfield as a regular binary integer, it may also be interpreted as a so-called Gray binary integer [3]. The advantages of Gray binary encoding, especially in connection with conformational analysis in torsion angle space, are explained in Chapter 2 (Section 5.2.1) in this thesis.

3.3 Operation

DENISE maintains a population of bitstrings that is allowed to evolve in a number of time steps, or "generations". Thereby, the number of bitstrings remains constant, say N.

First, all bits in the N bitstrings that comprise the population are set to a random binary value (0 or 1). In this form, the population enters the evolution cycle, which is briefly discussed next; for more details, the reader is referred to Chapter 2 (Section 3) in this thesis.

The first step in the evolution cycle involves decoding all bitstrings, and evaluation of these as such, using a suitable objective function. In this way, their respective utilities, or "fitnesses", are obtained.

The second step in the evolution cycle involves scaling the fitnesses in a way considered appropriate for the selection criterion used in the next step: reproduction. In DENISE, fitness scaling proceeds in three consecutive steps: (1) linear scaling, (2) normalization, and (3) sharing; for further details, the reader is referred to Chapter 2 (Section 6.1) in this thesis.

The third step in the evolution cycle is reproduction: N copies are made of N strings indicated, or "selected", in the population. DENISE uses *roulette selection*, which means that strings are selected probabilistically with an expected rate proportional to their scaled fitness. Accordingly, the new set of N bitstrings, called the temporary population, may be expected to feature a higher fitness, on average.

The fourth step in the evolution cycle is a random pairing of the bitstrings in the temporary population. Pairing serves the next step: recombination.

The fifth step in the evolution cycle is the application of a recombination procedure: a recombination operator is applied to all pairs of bitstrings assigned in the preceding step. The bitstrings in each pair of bitstrings exchange fractions in a way such that the loss of implicit information relevant for the search task is minimized; this principle is generally referred to as the *building block principle*. The recombination operator used in DENISE is **B_MX** (for 2 breakpoints), discussed in Chapter 2 (Section 5.2.2) in this thesis. Section 4.2 below provides an explanation, in terms of the building block principle, why **B_MX** may be regarded a good choice for the target problem.

The sixth step in the evolution cycle is the application of a mutation procedure: a mutation operator is applied to all bitstrings in the temporary population in order to invert the value of a small fraction of randomly selected bits. This serves to ensure that all parts of the search space remain reachable for the recombination operator, especially in the later stages of the search when population diversity is lost. The mutation operator used in DENISE is **B M**, discussed in Chapter 2 (Section 5.2.3) in this thesis.

The seventh step closes the evolution cycle: the population is replaced by the temporary population.

3.4 Evaluation

For the purpose of assigning fitness to the bitstrings comprising the population, DENISE employs a *cascaded bi-objective evaluation criterion*. More precisely, fitness assignment falls apart into two basic, sequentially calculated contributions – each from a separate, mono-objective evaluation criterion (i.e. objective function).

The first mono-objective evaluation criterion calculates, for each candidate conformation encoded in the population, an H-H distance table which is then compared to a table of known H–H distance constraints. The fitness contribution is proportional to the number of H–H distance constraints satisfied. This strategy allows finding a rough estimate of the globally optimal conformation through relatively little computational effort.

Only when all distance constraints are satisfied for a particular candidate conformation encoded in the population, the second mono-objective evaluation criterion is invoked for further refinement (hence *cascaded* bi-objective evaluation criterion). This criterion applies the aforementioned Bloch-Redfield model, and is computationally considerably more intensive due to many matrix multiplications involved. The fitness contribution is proportional to the goodness of fit between the calculated NOE table and the experimental NOE table.

Both contributions are multiplied by a separate weight factor (predefined by the user), and then added in order to obtain the total fitness.

When one of the weights is set to zero, this is interpreted by DENISE as an instruction to skip any effort in evaluating the corresponding contribution, i.e. to simply not use the mono-objective evaluation criterion concerned. (Setting both weights to 0, results in an error message.) In this way, DENISE can also be used in the two separate mono-objective evaluation modes, which, for the sake of brevity, are hereafter referred to by the self-explanatory terms *HH-only evaluation mode* and *NOE-only evaluation mode*, respectively.

Likewise, the cascaded bi-objective evaluation mode evaluation mode of DENISE is hereafter abbreviated as *HH/NOE evaluation mode*.

4 Experimental

This section presents a proof of principle application of DENISE, i.e. the aim is to obtain merely a first impression of the potential of genetic algorithms in conformational analysis of DNA.

4.1 Case study

The target DNA sequence adopted for DENISE in our pilot study, is the loop section in the DNA hairpin shown in Figure 2: $L_1L_2L_3L_4$. The rationale for choosing this hairpin is that, since recently, much is known about the three-dimensional structure of its loop from earlier studies [1], thus making it particularly easy to *simulate* experimental constraints. Simulations can be performed as follows: starting from realistic values for the set of conformational parameters describing the loop conformation, (1) H–H distance constraints are deduced in a straightforward way; (2) a NOE table is calculated using the Bloch-Redfield model. In turn, the rationale for simulation is that it allows verification of the results produced by DENISE, as the latter should be able

to *reconstruct* the known set of values for the conformational parameters. This experimental set-up makes sense at the present stage wherein still little is known about the relative viability of genetic algorithms as applied to conformational problems.

The simulated experimental constraints can be made even more realistic by:

1. Extending these constraints with similar constraints for a few close conformations that are less probable; this accounts for the fact that aqueous DNA manifests itself as a mixture of close conformations;
2. Adding a small amount of Gaussian noise to these constraints; this accounts for measurement errors;
3. Removing a small fraction from these constraints; this accounts for missing constraints, i.e. constraints that simply can not be determined for practical reasons.

Starting from realistic values for the set of torsion angles in $L_1L_2L_3L_4$, two sets of experimental constraints were realistically simulated according to the above guidelines:

- HH.$L_1L_2L_3L_4$ – containing all H–H distance constraints for $L_1L_2L_3L_4$, except for two constraints per nucleotide; these were randomly removed (guideline 3);
- NOE.$L_1L_2L_3L_4$ – containing a complete NOE table for $L_1L_2L_3L_4$; this table was changed as follows:
  - The table was extended with the NOE entries for four other conformations, assigned a ten times lower weight in calculating the fitness contribution (guideline 1); these minor conformations were selected randomly, subject to the condition that in any of these the value of any conformational parameter was not allowed to deviate more than a certain amount, say d, from the value of the corresponding torsion angle in the dominant conformation – whereby d was chosen as 1% of the size of the predefined value range, i.e. 3.6° in the case of torsion angles;
  - 1% Gaussian noise was added to all entries in the table (guideline 2).

HH.$L_1L_2L_3L_4$ was split into three logical parts: HH.L_1L_2, HH.L_2L_3, and HH.L_3L_4 – for L_1L_2, L_2L_3, and L_3L_4, respectively. Similarly, NOE.$L_1L_2L_3L_4$ was split into three logical parts: NOE.L_1L_2, NOE.L_2L_3, and NOE.L_3L_4 – again for L_1L_2, L_2L_3, and L_3L_4, respectively. In this way, it became possible to run DENISE separately for L_1L_2, L_2L_3, and L_3L_4, and then to use these results as a starting point for an "integral" refinement run dedicated to $L_1L_2L_3L_4$. The rationale for this strategy, called *problem partitioning*, is that an exponential reduction in the overall running time can be achieved; for further details, the reader is referred to Chapter 2 (Section 8.4) in this thesis.

A number of experiments were conducted, using in each case the appropriate set(s) of simulated experimental constraints. For convenience, these experiments are hereafter denoted according to the set(s) of experimental constraints used. For instance, an experiment wherein DENISE uses NOE.L_2L_3 as the set of experimental constraints, is denoted by NOE.L_2L_3. In that case, DENISE runs in NOE-only evaluation mode to perform a conformational analysis of L_2L_3. Similarly, when DENISE runs in HH/NOE evaluation mode to perform a conformational analysis of, say, L_1L_2 – i.e. when DENISE uses both HH.L_1L_2 and NOE.L_1L_2 as sets of experimental constraints – then the experiment is denoted by HH/NOE.L_1L_2.

Table 1 lists all experiments performed in this study. For further convenience, any set of experiments in a row or column of Table 1 is hereafter denoted by a name with a "wildcard". For instance, the experiments in the first row are denoted by *.L_1L_2; or, the experiments in the second column are denoted by NOE.*.

HH.L_1L_2	NOE.L_1L_2	HH/NOE.L_1L_2
HH.L_2L_3	NOE.L_2L_3	HH/NOE.L_2L_3
HH.L_3L_4	NOE.L_3L_4	HH/NOE.L_3L_4
HH.$L_1L_2L_3L_4$	NOE.$L_1L_2L_3L_4$	HH/NOE.$L_1L_2L_3L_4$

Table 1: Overview of experiments performed.

In each experiment, search was carried out in the full search volume; that is, for each of the conformational parameters the maximum relevant range of values was used.

4.2 Configuration

In all experiments, DENISE was configured as follows:

Population size	: 100
Selective reproduction	: fitness-proportional
Fitness scaling mode	: linear
Sensitivity	: 3.0
Sharing α	: 2.0
Sharing δ/L	: 0.1 (rel. Hamming distance)
Fraction elitism	: 0.05
Recombination mode	: **B MX** (2 breakpoints)
P_r	: 0.85
Mutation mode	: **B_M**
P_m	: 0.01
Encoding resolution	: 10 bits (1024 levels)
Decoding mode	: Gray binary

For a more detailed explanation of the configurational terminology, the reader is referred to Chapters 2 and 4 in this thesis. Here, we suffice with the following brief explanation of the choices that, as was empirically established, are critical for good performance:

- Sharing;
- Gray binary encoding;
- Recombination operator **B_MX**.

Other configurational factors turned out to be considerably less critical for good performance.

Sharing encourages the formation of stochastically stable sub-populations on different optima; for details, the reader is referred to Chapter 2 (Section 6.1.5) in this thesis. In conformational problems, this dispersal property is desirable for the following reasons. First, the experimental constraints are rarely so tight that they pinpoint a single conformation. Instead, they define a family of conformations, among which some may be biochemically nonsensical. Thus, sharing grants the expert biochemist the opportunity to choose the meaningful conformations among the set of all conformations encountered during a run, that satisfy the experimental constraints. Second, the best conformations may be found in non-global optima rather than in global optima, paradoxically. The reason for this lies in the fact that, in practice, artefacts in the experimental constraints can not be avoided, i.e. (as pointed out in Section 4.1 above): (1) the constraints apply to a mixture of conformations rather than to a single conformation; (2) the constraints are perturbed by noise; (3) the constraints are not tight, i.e. some constraints are missing.

The virtue of Gray binary encoding resides in the fact that it avoids Hamming gaps in the binary search space, and that advantage can be taken of its cyclic nature, as most conformational parameters are periodic; for details, the reader is referred to Chapter 2 (Section 5.2.1) in this thesis.

The choice of the **B_MX** operator is explained by the fact that, starting from reliable general knowledge about the target problem, an educated guess can be made at an encoding underlying the operator, such that the aforementioned building block principle is obeyed. This particular encoding involves, more precisely, bitstrings in which the bitfields for any pair of conformational parameters are placed closeby if it is known that these parameters are strongly correlated (in the sense that large fitness contributions occur only when both parameters adopt certain values simultaneously), because this maximizes the chance that both bitfields are propagated together from a parent string to a child string – hence that important information is preserved. Such a sequence of optimally placed bitfields is called a *connected sequence* of bitfields; for further details, the reader is referred to Chapter4 (Section 4.1.2, Frame 2) in this thesis. As for establishing the extent of correlation between the conformational parameters, the following can be remarked.

Since correlations between the conformational parameters should be ascribed to spatially interacting molecular components, it is reasonable to assume, in first approximation, that the correlation between two conformational parameters is inversely proportional to the distance between the molecular components they refer to; this distance may be defined in several ways, e.g. simply as the minimal number of separating chemical bonds. In DENISE, these considerations have lead to an educated guess at a connected sequence of bitfields according to (Figure 1):

$$\cdots \zeta_{i-1}\ \alpha_i\ \beta_i\ \gamma_i\ \chi_i\ \nu_{m,i}\ P_i\ \epsilon_i\ \zeta_i\ \alpha_{i+1} \cdots$$

4.3 Software details

The routines in any genetic algorithm basically comprise two parts: a domain-dependent part (concerning the problem representation and the evaluation criterion) and a domain-independent part (concerning the evolutionary search heuristics). By "domain-independent" routines are

meant routines that can be used for other problem domains as well; in our application, these were obtained from the software library GATES [7, 8]. In conformity with this library, the domain-dependent routines and data structures were programmed in C (ANSI standard) for speed and portability. The integration of all routines resulted in DENISE. Two versions of DENISE were created: one runs under UNIX on SUN computers, and the other under MS DOS on 80486 computers; the former version was used for this study.

5 Results and discussion

This section starts with a global summary of the results obtained for all experiments listed in Table 1. Thereafter, the results for experiment NOE.L_2L_3 are presented in detail.

5.1 Global summary of results

In all experiments, the best bitstring encountered during a run, turned out to represent an acceptable conformation. Thus, in general, it was possible to reconstruct the conformation of $L_1L_2L_3L_4$ (Figure 2).

In most experiments, convergence was observed after approximately 2500 generations. However, in order to improve the chances of finding an even more acceptable conformation, runs were aborted only after 4500 generations, i.e. 450,000 evaluations (since the population size was 100). This still comes down to an extremely tiny sample of the conformational space (see Section 3.1 above). Moreover, reasonable time limits were not exceeded: for experiments NOE.L_iL_{i+1} ($i = 1, 2, 3$), the 450,000 evaluations come down to a running time of approximately 50 hours (real-time on the computer used for all experiments). The running time for experiments HH.L_iL_{i+1} was considerably smaller – by a factor 20, approximately – but the results were less precise, as was assessed as follows.

Both the HH.* experiments and NOE.* experiments were replicated a number of times in order to compare the spread in results. It turned out that in any HH.* experiment, the spread in results was larger than the spread in results in the NOE.* experiment involving the same DNA sequence. However, the difference was not dramatic, certainly not as dramatic as the difference in running times. This indicates that DENISE's HH-only evaluation mode may suffice in some cases, depending on the precision pursued by the practitioner – who, of course, realizes that the practically feasible precision depends on the amount of H–H distance constraints available.

	Target	Predicted
γ_{L_2}	55.0	54.9
ϵ_{L_2}	240.0	240.7
ζ_{L_2}	235.0	234.7
χ_{L_2}	217.0	217.1
P_{L_2}	178.0	177.9
ν_{m,L_2}	43.0	43.0
α_{L_3}	299.0	299.5
β_{L_3}	198.0	197.4
γ_{L_3}	78.0	77.8
χ_{L_3}	187.0	186.9
P_{L_3}	198.0	197.9
ν_{m,L_3}	41.0	41.2

Table 2: Target values of conformational parameters for the dominant conformation (experiment NOE.L_2L_3), and values predicted by DENISE. (Values are in degrees.)

In all experiments, a considerable gain in convergence time (number of generations) could be achieved by using less than 10 bits for the bitfields. However, as expected, this lead to a loss in precision (and accuracy), i.e. to a larger spread in results when a particular experiment was replicated a number of times. Nonetheless, preliminary results indicate that the use of small bitfields can be rewarding as part of a strategy wherein different runs of DENISE are performed sequentially, as follows: the result of a particular run is used by the next-scheduled run for refinement; the refinement is achieved through expansion of bitfields by one (or a few) bits. This strategy is an example of a sequential self-hybridization strategy, called iterated search grid expansion; for further details, the reader is referred to Chapter 2 (Section 8.3.3) in this thesis.

5.2 Results for NOE.L_2L_3

Table 2 lists target values of the relevant conformational parameters for the assumed dominant conformation of L_2L_3, and the values predicted by DENISE for these parameters. It follows that all predictions are within 0.5° deviation from the target values. Hence, it may be concluded that

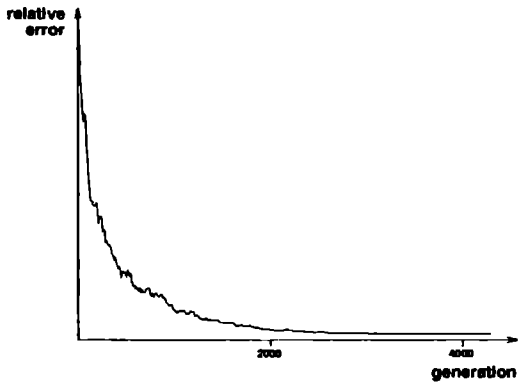

Figure 3: Evolution of root-mean-square error of fit for the best conformation (experiment NOE.L_2L_3).

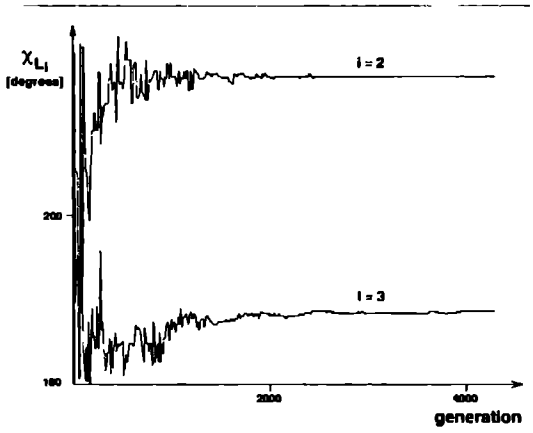

Figure 4: Evolution of predicted values of torsion angles χ_{L_2} and χ_{L_3} for the best conformation (experiment NOE.L_2L_3).

DENISE is able to reconstruct the target values within "experimental" error.

Figures 3, 4, and 5 illustrate several important aspects of the evolution process; comment is provided in their captions.

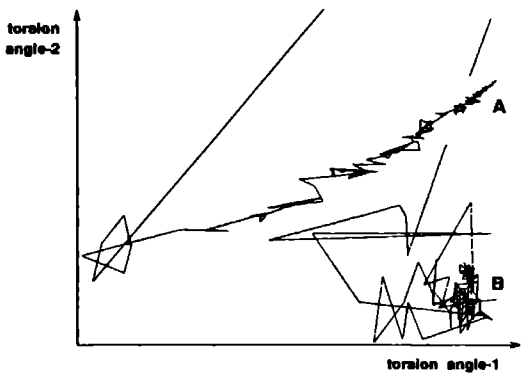

Figure 5: Search trajectory of the best conformation in autoscaled, projected conformational space (experiment NOE.L_2L_3), revealing the extent of correlations between torsion angles (Section 4.2). A: between ζ_{L_2} and β_{L_3} (strongly correlated torsion angles). B: between χ_{L_2} and χ_{L_3} plane (weakly correlated torsion angles).

6 Conclusions and outlook

This paper presented DENISE: a genetic algorithm for conformational analysis of DNA. DENISE is designed for on-line analysis of two kinds of experimental constraints: H–H distance constraints and 2D-NOE spectra, either separately or combined. From a carefully set up case study, it follows that DENISE searches the conformational space in an efficient and robust way, as it finds, within reasonable time, meaningful conformations that satisfy all constraints within experimental error. However, as a consequence of the probabilistic nature of the search, results are not exactly reproducible. A viable strategy to obtain preciser conformations probably resides in local search applied to the result of DENISE. Thereby, the local search may use another objective function – e.g. one based on an assumed force field so that energy minimization will be carried out, as in restrained molecular dynamics. Such exploitation of traditionally used, i.e. already available, techniques is attractive in that it minimizes the total effort put into the design of the more powerful, hybrid searching system. Moreover, as the traditionally used technique gains a wider scope of application through such hybridization, it may stay in use for a longer time, hence pay back more.

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research (SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007.

We wish to thank Prof. Dr. C.W. Hilbers and Dr. F.J.M. van de Ven from the Laboratory for Biophysical Chemistry (Catholic University of Nijmegen) for fruitful discussions and other kind co-operation.

References

[1] Blommers, M.J.J. *Aspects of Loopfolding in DNA Hairpins.* PhD thesis, Catholic University of Nijmegen, 1990.

[2] Blommers, M.J J., Lucasius, C.B., Kateman, G., and Kaptein R. Conformational analysis of a dinucleotide photodimer with the aid of the genetic algorithm. *Biopolymers*, 32:45–52, 1992.

[3] Caruana, R.A. and Schaffer, J.D. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In Laird, J , editor, *Proceedings of the Fifth International Conference on Machine Learning*, page 153, San Mateo, CA, 1988 Morgan Kaufmann

[4] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] Holland, J.H. *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975. Revised print· MIT Press, Cambridge, MA, 1992.

[6] Lucasius, C B., Blommers, M J J , Buydens, L.M C., and Kateman, G A genetic algorithm for conformational analysis of DNA. In Davis, L , editor, *Handbook of Genetic Algorithms*, pages 251–281, New York, NY, 1991. Van Nostrand Reinhold. Chapter 18.

[7] Lucasius, C B and Kateman, G. GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms. Part 1. General perspective. *Computers & Chemistry*, 1993 Accepted.

[8] Lucasius, C B and Kateman, G. GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 2 Toolbox description *Computers & Chemistry*, 1993. Accepted

[9] Macura, S and Ernst, R.R. Elucidation of cross relaxation in liquids by two-dimensional NMR spectroscopy. *Molecular Physics*, 41(1) 95, 1980

[10] Metzler, W.J , Hare, D R , and Pardi, A. Limited sampling of conformational space by the distance geometry algorithm· implications for structures generated from NMR data *Biochemistry*, 28 7045, 1989

[11] Saenger, W. *Principles of Nucleic Acid Structure.* Springer-Verlag, New York, NY, 1984

[12] van de Ven, F.J.M. and Hilbers, C W Nucleic acids and nuclear magnetic resonance. *European Journal of Biochemistry*, 178 1, 1988. Review.

CHAPTER 7

Conformational Analysis of a Dinucleotide Photodimer with the Aid of the Genetic Algorithm

Contents

Conformational Analysis of a Dinucleotide Photodimer with the Aid of the Genetic Algorithm

MARCEL J. J. BLOMMERS,[1] CARLOS B. LUCASIUS,[2] GERRIT KATEMAN,[2] and ROBERT KAPTEIN[1]

[1]Bijvoet Center, University of Utrecht, Padualaan 8, 3584 CH Utrecht; and [2]Department of Analytical Chemistry, Faculty of Sciences, University of Nijmegen, Toernooiveld, 6525 ED Nijmegen, The Netherlands

SYNOPSIS

The solution structure of the photodimer *cis,syn*-dUp[ ]dT is derived with the aid of the genetic algorithm. The conformational space available for the molecule is sampled efficiently using the computer program DENISE and tested against a set of constraints available from nmr experiments. The dominant conformation in solution found with this approach can be described by the following combinations of sugar–phosphate backbone torsion angles: $\epsilon(t)$, $\zeta(t)$, $\alpha(+)$, $\beta(-ac)$, and $\gamma(t)$. The conformation of the sugars and glycosidic torsion angles are S type and *syn*, respectively. The cyclobutane ring and pyrimidines are puckered. In addition, other conformations that exist in equilibrium with the first are found. It is concluded that the cyclobutane-pyrimidine system is rigid, whereas the sugar–phosphate backbone is flexible. The solution structures are compared with the crystal structure of the strongly related cyano-ethyl ester of *cis,syn*-dTp[ ]dT.

INTRODUCTION

Nuclear magnetic resonance is widely used for structural analysis of molecules. Cross-peak intensities in nuclear Overhauser enhancement (NOE) spectra, *J*-coupling constants, and chemical shifts depend on the conformation of a molecule. Therefore, these experimentally obtainable data are generally used for structure determination of molecules. The computational methods mainly used so far for structure determination based on nmr data include restrained molecular dynamics and distance geometry and variants thereof. Recently, a new approach was developed that makes use of the so-called genetic algorithm. The concept of this algorithm was first introduced by Holland.[1] The main idea of the genetic algorithm is founded upon the basic concepts of Darwin's evolution theory, i.e., generalizations of mutation, recombination, selection, and survival of the fittest. This has resulted, thus far, in a number of widely varying applications[2,3] such as parameter optimization, network design, robot trajectory generation, and aircraft design. The computer program DENISE has been developed to use this approach in conformational analysis of nucleic acid molecules. The program and tests with theoretical datasets has been described earlier.[4,5] Here, its use will be presented in relation to structural analysis of the photodimer *cis,syn*-dUp[ ]dT from experimental nmr data. The conformation of this molecule has been studied before by nmr in combination with distance-geometry and restrained molecular dynamics.[6] Although the previously calculated structures of the molecule are consistent with most nmr data, not all the vicinal *J* couplings are properly accounted for. This and the fact that the molecule exists in an equilibrium of multiple conformations make it an interesting candidate to study with the genetic algorithm, since this method may provide a more exhaustive search of conformational space. We show here that this approach indeed yields conformations that satisfy all nmr constraints.

The *cis,syn*-dUp[ ]dT is one of the major photoproducts of the dinucleotide dCpdT. Upon uv irradiation of the latter molecule a cyclobutane ring is formed, consisting of the C5 and C6 atoms of both pyrimidines. The *cis,syn* product deaminates spontaneously, which yields the molecule sketched in Figure 1. The pyrimidine photodimers are the major products of the photoinduced damage of DNA. Thus far, photoproducts of dinucleotides have been stud-

Biopolymers, Vol 32, 45–52 (1992)
 CCC 0006-3525/92/010045-08$04.00

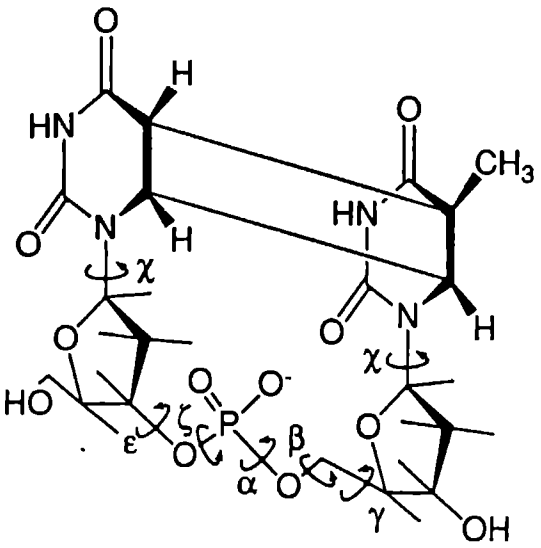

Figure 1. Structure formula of the *cis,syn*-dUp[ ]dT. The conformation of the most important torsion angles are indicated.

ied with nmr,[6] x-ray crystallography,[7] and molecular mechanics calculations.[8] The present nmr-based structures will be compared with the results from crystallography.

EXPERIMENTAL

Conformational Parameters of the Photodimer

Assuming that bond distances and bond angles are constant, the conformation of a molecule can be described by a set of torsion angles. As shown in Figure 1, the conformations of *cis,syn*-dUp[ ]dT are described by the phosphate backbone torsion angles, the sugar pucker, the glycosidic torsion angles χ, and those describing the pyrimidine and cyclobutane ring systems. The conformation of the deoxyribose sugars are described by two parameters: the pseudorotational phase angle P and the pucker amplitude ϕ_m. Thus, apart from the bases and cyclobutane rings, the conformations of the photodimer are described by 11 parameters, i.e., $\chi(U)$, $P(U)$, $\phi_m(U)$, ϵ, ζ, α, β, γ, $P(T)$, $\phi_m(T)$, and $\chi(T)$ (cf. Figure 1). The pyrimidine bases are assumed planar (in a first approximation). The C5(U)-C5(T) and C6(U)-C6(T) bonds forming the cyclobutane ring are treated as short interatomic distances. A more realistic conformation of this part of the molecule is obtained after energy refinement (vide infra).

Genetic Algorithm

The general outline of the genetic algorithm is sketched in Figure 2. Central in this method is a pool of conformations that are encoded in binary strings. A bitstring is partitioned in equally sized segments called bitfields, each of which encodes a torsion angle of the molecule. In our case the bitfields are treated as Gray integers rather than normal integers.[9] Gray integers have the property that a change in any of its bits yields a decoded real value that is close to the original one. Another advantage of Gray coding is that it is cyclic; this implies that for torsion angles the values 0 and 360 are close to each other in Gray space.

In the first stage, the torsion angles of the "starting structures" are randomly chosen within torsion angle bounds. These bounds may follow from an analysis of coupling constants, which reduces the search space considerably. Using these initial torsion angles a population of conformations is generated. Subsequently the conformations are tested against the experimental distance constraints, defined as upper and lower bounds. The violations between the theoretical and the experimental data are used to calculate the individual *fitness* or performance values. At this stage the first generation of a population of bitstrings, designated as *chromosomes*, is obtained. Each individual (chromosome) of the population has a fitness value. Thus, the individuals in the population represent a pool of candidate solution structures and their fitness factors. Now we can apply the evolution mechanism according to Darwin's laws: after mutation, selection, and recombination, the bitstrings representing the fittest individuals will

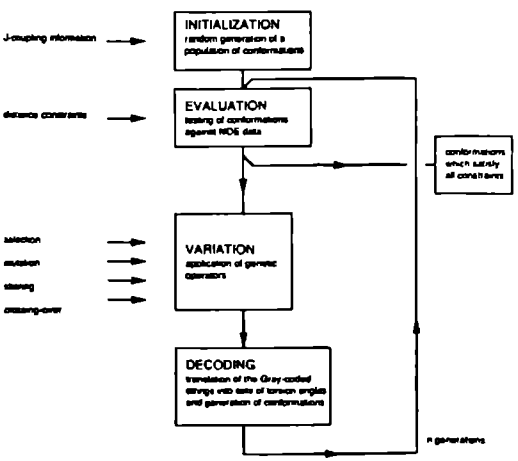

Figure 2. Flowchart of the genetic algorithm.

survive. In other words, after n generations the conformations will converge to those that give an improved fit to the applied constraints. In the mutation stage we apply four genetic operators to our system. Only the best individuals are selected (selection), and one or more of the bits in the chromosome are switched (mutation). Pairs of chromosomes are recombined, i.e., exchange of part of the chromosomes takes place (crossing over). Finally, the concept of sharing is applied.[10] In short, this option reduces the fitness of particular conformations to prevent the generation of twins. If this is not done, a number of almost identical conformations could occur for a considerable part of the conformation pool. The conformations obtained from the mutated chromosomes are then evaluated and this procedure is repeated for several generations.

The program DENISE, which incorporates the approach outlined above, is written in the language C and runs on hardware that supports ANSI-standard C compatible compilers, e.g., a SUN SPARC workstation. The analysis described in this paper, in which 100 individuals during 25 generations were evaluated, was achieved in about 3 central processing unit minutes.

Energy Refinement

The structures generated by DENISE were further refined with molecular mechanics calculations. The program DISCOVER,[11] which runs on a Silicon Graphics IRIS workstation, was used with the all-atom force field of AMBER.[12] For the modified part of the photodimer, the C5 and C6 atoms were treated as standard sp^3 carbons.[8] The proton and methyl substituents of the cyclobutane ring were placed on the sp^3 C5 and C6 atoms, with the proper chirality. A hydrated counterion is placed near the negatively charged phosphate.[13] A distance-dependent dielectric constant ($\epsilon = 1/R$) was used. The structure was minimized 100 steps with steepest descent and subsequently with conjugate gradient optimization, until convergence of the energy was achieved. This procedure was found to be necessary to optimize the covalent bonds at the cyclobutane region of the molecule.

RESULTS

The rotamer populations for the *cis,syn*-dUp[ ]dT deduced from coupling constants in our previous investigation[6] are collected in Table I. The H5′ and H5″ resonances were only pairwise assigned, and therefore the J-coupling data refer to either a 75% *gauche*(−)/25% *gauche*(+) distribution of the torsion angle γ, or to a 75% *trans*/25% *gauche*(+) distribution depending on the stereospecific assignment. It can be inferred from Table I that more than one conformation exist in solution. The interchange of these conformations is fast on the nmr time scale. The distance constraints from NOEs found previously[6] and used in this work are listed in Table II. It is noteworthy that the H1′(U) is found to be close to H5′(T) and that H6(T) is close to H1′(U), i.e., the distances between these protons are less than 3–4 Å, at least in one of the dominant conformations.

Table I Interpretation of NOE Cross Peaks and J-Coupling Constants for *cis,syn*-dUp[ ]dT[a]

Torsion Angle[a]	Experimentally Obtained Results
Sugar(T)	88% S type, 12% N-type
χ(T)	*syn*[b]
ϵ	*gauche*(−) and/or *trans*
ζ	Not known
α	Not known
β	60% *trans*
γ	75% *gauche*(−), 25% *gauche*(+) or 75% *trans*, 25% *gauche*(+)
Sugar(U)	63% S type, 37% N type
χ(U)	*syn*

[a] The torsion angles are indicated in Figure 1. The analysis of the sugar conformation was carried out with PSEUROT. The conformation of χ was estimated semiquantitatively on the basis of cross-peak intensities in a NOESY spectrum. The population of the various rotamers are estimated with sum rules. [b] With *syn* conformations of χ are meant $-90° < \chi < 90°$; this includes high *anti* and *syn* conformations.

As pointed out above, the molecule is not conformationally pure. Therefore in the first stage, we aimed at determining the major conformation. Inspection of Table I reveals that according to the nmr data several combinations of phosphate backbone torsion angles, e.g., $\gamma(-)$ or $\gamma(+)$, $\beta(t)$, $\epsilon(t)$, or $\epsilon(-)$ may occur. Both sugars are S type and the glycosidic torsion angles are *syn*. Using this information we performed calculations with DENISE and tested the conformations against distance constraints. In the first calculation γ is constrained to the *gauche*(−) region, assuming one of the possible individual assignments of the H5′ and H5″ resonances. The torsion angle ranges are given in Table II. The calculation is performed with a pool of 100 conformations. A combination of all genetic operators is used; the parameters are collected in Table II. After 25 generations convergence is reached: a conformation is found which satisfies all constraints. Figure 3 shows

Table II Experimental Data and Parameters Used as Input for the Program DENISE (cf. Figure 2)

Distance Constraints (Å)		
Proton Pair	**Lower Bound**	**Upper Bound**
H6(U)-H2′(U)	2.0	3.0
H6(T)-H2′(T)	1.8	2.5
C5(U)-C5(T)	1.4	3.0
C6(U)-C6(T)	1.4	3.0
H1′(U)-H5′(T)	1.8	4.0
H1′(U)-H6(T)	1.8	4.0
Other	1.8	—

J-Coupling Information		
Conformational Parameter	**Lower Bound**	**Upper Bound**
χ(U)	−90°	90°
P(U)	120°	200°
ϕ_m(U)	32	44
ε(U)	120°	360°
ζ(U)	0°	360°
α(T)	0°	360°
β(T)	120°	240°
γ(T)	240°	360°
P(T)	120°	200°
ϕ_m(T)	32	44
χ(T)	−90°	90°

Genetic Operators	
Parameter[a]	**Value**
Pool size	100
Mutation probability	0.03
Pairing probability	0.7
Sharing probability	0.1
Sharing parameter σ	0.5
Sharing parameter α	0.5
Sharing amplitude	0.5

[a] The genetic parameters are defined as follows: pool size represents the number of bitstrings in the population. Mutation probability is the probability by which each bit in the population flips value (1 to 0; 0 to 1). Pairing probability is the probability by which selected bitstring pairs perform crossing over. Sharing stimulates the heterogeneity of the population, its parameters are explained elsewhere.[10]

the performance of DENISE. Along the vertical axis the violation from the target distances in arbitrary units is given as a function of the generation number. The conformation is subjected to an energy minimization and the resulting conformation is presented in Figure 4A. The torsion angles of this conformer is given in Table III. The conformation corresponds with all nmr data: The torsion angles of the backbone are γ(−), β(t), and ε(t). Because γ adopts a *gauche*(−) conformation, the H5′ and H5″ protons point inside the cyclic 14-membered ring, formed by sugars, phosphate backbone, and pyrimidines. The latter observation gives rise to a short H5′(T)-H1′(U) distance, which is responsible for the observed NOE. The torsion angles of the cyclobutane and the pyrimidines are also given in Table III (conformation A). Neither the pyrimidines nor the cyclobutane ring is flat. The latter is nicely confirmed by the *J* couplings between the protons attached to the four-membered ring.

Next, a calculation is performed in which the reversed assignment of the H5′ and H5″ resonances is assumed. This resulted in the conformation depicted in Figure 4B. The torsion angles are collected in Table III. This model also explains the observation of a NOE between H5″(T) and H1′(U). Thus,

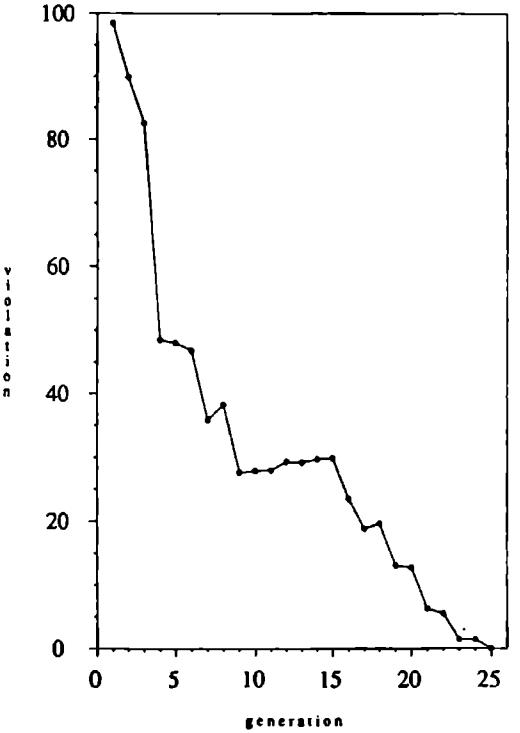

Figure 3. Performance of the genetic algorithm using the program DENISE, applied to find conformation A.

Table III Torsion Angles and Energy of the Refined Structures[a] of *cis,syn*-dUp[ ]dT and the Crystal Structure of the Cyanoethyl Ester of *cis,syn*-dTp[ ]dT[c]

Torsion Angle	A	B[b]	C	D	X-Ray
β(U)	53°	51°	53°	47°	—
γ(U)	62°	62°	60°	64°	295°
δ(U)	136°	148	137	151	159
χ(U)	36°	52°	41°	53°	61°
P(U)	151°	158°	154°	153°	175°
ϕ_m(U)	37	44	39	44	40
ϵ(U)	180°	193°	174°	283°	207°
ζ(U)	231°	196°	248°	136°	222°
α(T)	28°	73°	284°	273°	59°
β(T)	173°	244°	157°	113°	235°
γ(T)	285°	197°	51°	37°	189°
P(T)	216°	154°	139°	108°	32°
ϕ_m(T)	38	44	43	47	42
χ(T)	310°	285°	298°	300°	256°
δ(T)	167°	146°	135°	103°	81°
ϵ(T)	302°	297°	175°	64°	270°
ϕ_1[c]	23°	22°	22°	16°	21°
ϕ_2	336°	338°	338°	344°	339°
ϕ_3	23°	22°	22°	16°	21°
ϕ_4	336°	338°	338°	344°	339°
Ψ_1(U)[d]	23°	8°	18°	3°	22°
Ψ_2(U)	0°	4°	0°	11°	358°
Ψ_3(U)	351°	1°	356°	354°	355°
Ψ_4(U)	355°	343°	350°	349°	353°
Ψ_5(U)	25°	26°	25°	23°	23°
Ψ_6(U)	326°	337°	330°	340°	328°
Ψ_1(T)	1°	6°	1°	354°	4°
Ψ_2(T)	18°	8°	12°	3°	8°
Ψ_3(T)	352°	0°	359°	13°	5°
Ψ_4(T)	342°	341°	340°	336°	333°
Ψ_5(T)	34°	31°	31°	21°	37°
Ψ_6(T)	333°	334°	338°	354°	333°
Energy (kcal)	−87	−95	−97	−96	—

[a] A–D represent four refined structures of *cis,syn*-dUp[ ]dT that were found with the genetic algorithm (see text).
[b] Conformer B is identified as the dominant conformation in solution.
[c] The endocyclic torsion angles of the cyclobutane ring are numbered as follows: ϕ_1 = C5(U)-C5(T)-C6(T)-C6(U); ϕ_2 = C5(T)-C6(T)-C6(U)-C5(T), etc.
[d] The endocyclic torsion angles of the pyrimidines are numbered as follows: Ψ_1 = C6-N1-C2-N3; Ψ_2 = N1-C2-N3-C4, etc.

at this point it is not yet possible to choose between model A and model B. When the calculated energy is compared, however, the energy of conformer B is 8 kcal lower than that of conformer A.

In an attempt to generate conformations that correspond to the minor populated ones, we removed the H5′(H5″)-H1′ distance constraint, in order to allow other conformations than those with $\gamma(-)$. Two conformations, designated C and D, were found and are presented in Figure 4. Both conformations have the torsion angle γ in the *gauche*(+) domain. In conformation C the ϵ is *trans*, while in conformation D the ϵ adopts a *gauche*(−) conformation. The occurrence of a conformation, which has an $\epsilon(-)$ conformation, is corroborated by the presence of a long-range $J_{2'P}$ coupling of −1.4 Hz. In addition to conformations A–D, were also conformational variants found, which have the 3′ sugar of the thymidine part of the molecule in a N-type conformation.

DISCUSSION

The Genetic Algorithm

Up to now distance geometry (DG) and restrained molecular dynamics (RMD) have been widely used for structure determination of molecules. Often a structure is generated by DG embedding and subsequently refined by restrained energy minimization, simulated annealing, or restrained molecular dynamics. The success of these strategies strongly depends on a proper sampling of the conformational space. The optimization strategy that follows after

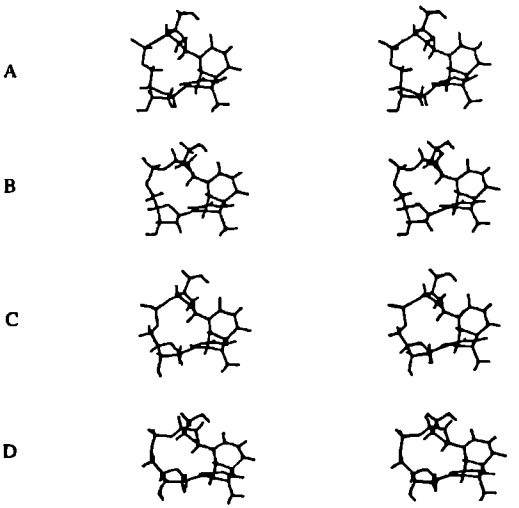

Figure 4. Stick plots in stereo view of the conformations of *cis,syn*-dUp[ ]dT, which are found by DENISE and subsequently energy minimized using the AMBER force field. Torsion angles are collected in Table III.

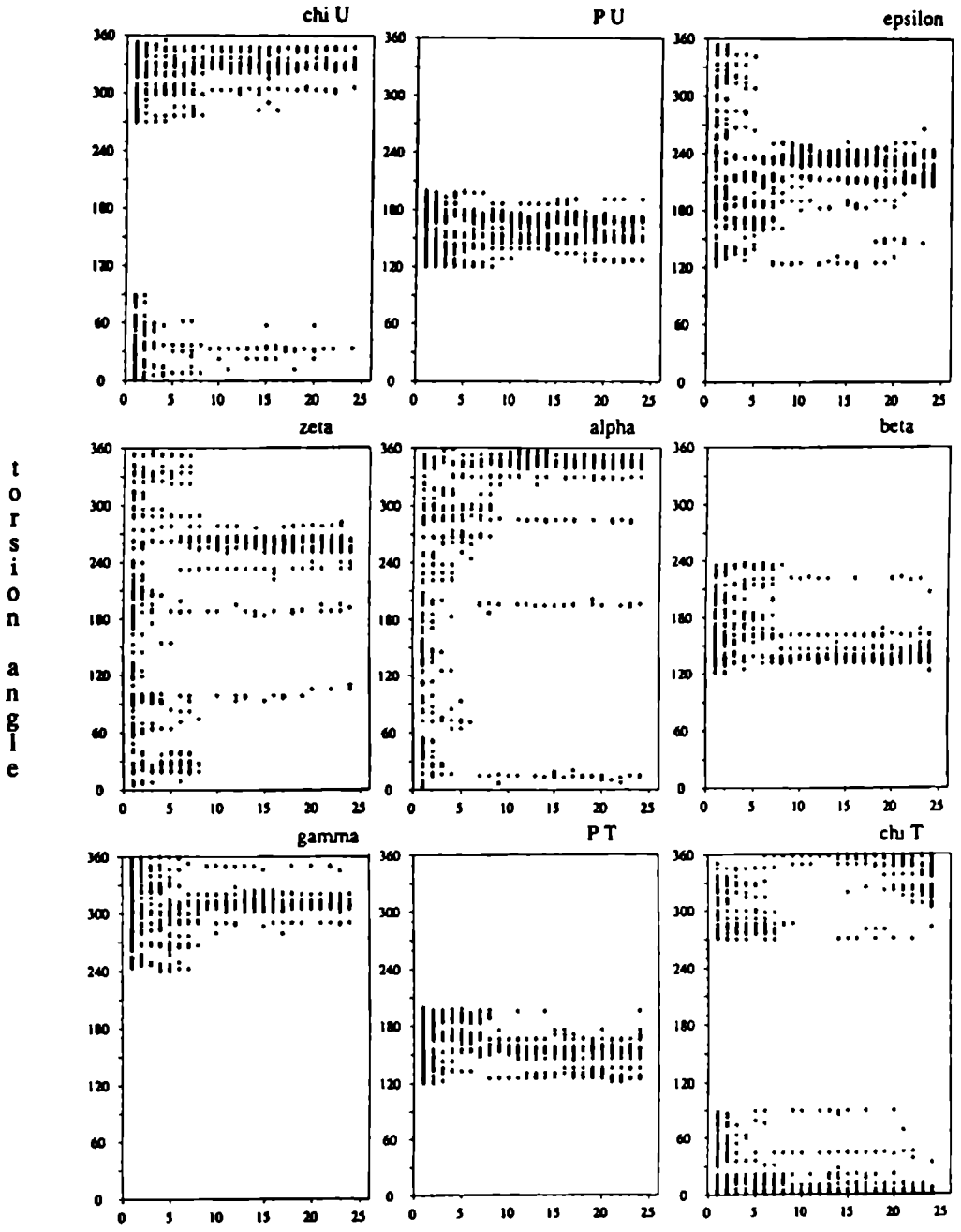

Figure 5. Distribution of the torsion angles of the conformers during the optimization of the gene pool (case A). Each occurrence of a torsion angle value is indicated with a dot. During the first generations the sampling is spread evenly over the conformational space, while at the end the sampling is spread around one or more solutions.

the structure generation may be complicated by the fact that many local minima may exist in a vast solution space. This problem is alleviated somewhat by the use of molecular dynamics, which has better convergence properties by being able to pass over small barriers.

It is interesting to see how these problems can be tackled with the aid of the genetic algorithm. The sampling of the conformational space can be envisioned by plotting the occurrence of values of torsion angles in the conformation pool as a function of the evolution time (generation number). In this way the sampling of the conformational space can be inspected. This is shown in Figure 5 for the most important structural parameters, i.e., torsion angles and pseudo-rotational phase angles, in the calculation that resulted in conformer A. During the first generations, the conformational space is sampled throughout within the torsion angle bounds, while at the end the conformational space around the solution is sampled predominantly. Moreover, it can be viewed from Figure 5 that other combinations of torsion angles are also further optimized during the evolution process. As a logical consequence, the aforementioned local minimum problem is circumvented, because several solutions converge side by side (cf. Figure 5). In conclusion, the current implementation of the algorithm exhibits good explorative sampling as well as exhaustive sampling around the local minima.

The Solution Structures of *cis,syn*-dUp[ ]dT

The present investigation of the solution structure of the *cis,syn*-dUp[ ]dT gives an improved conformation of the molecule with respect to a previous study, especially concerning the conformation of the phosphate backbone. An impression of the similarities of the four conformers is obtained when the four carbons of the cyclobutane ring are superimposed. This is shown in Figure 6 and it appears that the conformation of the pyrimidine rings as well as the conformation of the cyclobutane does not change from one structure to the other. It can be inferred from Figure 6 that it is the sugar–phosphate backbone, which is the flexible part of the molecule. Several combinations of the torsion angles of the phosphate backbone are possible, and it appears that the distance between the two C1′ atoms and the two vectors along the C1′-N1 bond are fairly constant.

It is interesting to compare the conformation around the cyclobutane ring with that found in the crystallographic study. The lower part of Table III gives all the relevant torsion angles of the cyclobutane–pyrimidine system of the three solution structures and that of the x-ray structure of the strongly related cyano-ethyl ester of *cis,syn*-dTp[ ]dT. These torsion angles are seen to be in good agreement. It was suggested by Hruska et al. that the conformation of this part of the molecule could be flexible in solution. Evidence for pseudorotation of the four-membered ring, however, cannot be obtained from the available *J* couplings because in the two possible puckers the endocyclic torsion angles give rise to almost the same coupling constants. This study shows that the conformation of the cyclobutane ring and the attached pyrimidines is roughly the same in the four conformers, and therefore is probably rigid. The cyclobutane is puckered with a dihedral angle of 157°. The U has a $^{6}H_{1}$ half-chair conformation, while the T can be characterized by a $_{6}H^{5}$ half-chair conformation.

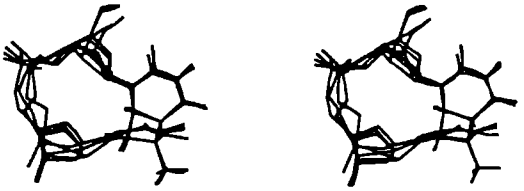

Figure 6. Superposition of the four conformers, referred to as A–D in Figure 4 and Table III. Here, the four C atoms of the cyclobutane ring are superimposed.

A comparison of the flexible part of the solution structures with that of the crystal structure is also interesting. As was pointed out before, conformation B is more favorable than A from an energetic point of view. Comparison of the two conformations with the crystal structure corroborates this observation. It is concluded that conformation B corresponds to the dominant structure in solution. Thus, all phosphate backbone torsion angles of the crystal structure are preserved in solution as well (cf. Table III). However, the sugar conformation of the 3′T sugar is switched in solution from a N- to a S-type conformation. In addition, slight changes of the glycosidic torsion angles are apparent, as well as some of the sugar–phosphate backbone torsion angles, which may be a consequence of the change in sugar pucker.

REFERENCES

1. Holland, J. H. (1975) *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.
2. Goldberg, D. E. (1989) *Genetic Algorithms in Search,*

Optimization and Machine Learning, Addison-Wesley, Reading, MA.
3. Davis, L. (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
4. Lucasius, C. B., Blommers, M. J. J., Buydens, L. M. C. & Kateman, G. (1991) in *Handbook of Genetic Algorithms*, Davis, L., Ed., Van Nostrand Reinhold, New York, pp. 251–281.
5. Lucasius, C. B., Blommers, M. J. J., Werten, S., van Aert, A. H. J. M. & Kateman, G. (1990) in *Proceedings of the Workshop Parallel Problem Solving from Nature*, Dortmund October 1–3, in press.
6. Koning, T. M. G., van Soest, J. J. G. & Kaptein, R. (1991) *Eur. J. Biochem.* **195**, 29–40.
7. Hruska, F. E., Voituriez, L., Grand, A. & Cadet, J. (1986) *Biopolymers* **25**, 1399–1417.
8. Rao, S. N., Keepers, J. W. & Kollman, P. A. (1984) *Nucleic Acids Res.* **12**, 4789–4807.
9. Caruana, R. A. & Schaffer, J. D. (1988) in *Proceedings of the Fifth International Conference on Machine Learning*, Laird, J., Ed., Morgan Kaufmann, San Mateo, CA, pp. 153–161.
10. Deb, K. & Goldberg, D. E. (1989) in *Third International Conference on Genetic Algorithms*, Schaffer, J. D., Ed., Morgan Kaufmann, San Mateo, CA, pp. 42–50.
11. Biosym Technologies, Inc. (1989) *Discover 2.5*, San Diego, CA.
12. Weiner, S. J., Kollman, P. A., Nguyen, D. T. & Case, D. A. (1986) *J. Comp. Chem.* **7**, 230–235.
13. Singh, U. C., Weiner, S. J. & Kollman, P. A. (1985) *Proc. Natl. Acad. Sci. USA* **82**, 755–759.

Received June 6, 1991
Accepted September 24, 1991

CHAPTER 8

Multicriteria Target Vector Optimization of Analytical Procedures Using a Genetic Algorithm. Polyoptimization of the Photometric Calibration Graph of Dry Glucose Sensors for Quantitative Clinical Analysis

Contents

Multicriteria target vector optimization of analytical procedures using a genetic algorithm

Part II. Polyoptimization of the photometric calibration graph of dry glucose sensors for quantitative clinical analysis

D. Wienke and C. Lucasius

Laboratory for Analytical Chemistry, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands)

M. Ehrlich

Medizinische Akademie "Carl Gustav Carus" Dresden, Institut für Biochemie, Karl-Marx-Strasse 3, D(O)-8080 Dresden (Germany)

G. Kateman

Laboratory for Analytical Chemistry, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands)

(Received 1st June 1992; revised manuscript received 10th September 1992)

Abstract

The target vector criterion as the vector distance between a set of desired responses and a set of really obtained responses of a multivariate function was combined with the genetic algorithm to improve simultaneously six properties of a biochemical test strip for human blood glucose determination as a function of twelve chemical and technological parameters. The advantage of the genetic algorithm in comparison with other search techniques became obvious for the search in this twelve-dimensional variables space with up to seven bit resolution, i.e., up to 1.93^{25} search positions. The results obtained by target vector optimization on the basis of the genetic search technique were critically compared with the results obtained by a prediction with classical and non-linear partial least-squares regression and realized in laboratory and industrial verification experiments. In this way advantages and disadvantages of deductive and inductive polyoptimization strategies could be discussed theoretically and with respect to experimental results.

Keywords: Optimization methods; UV–Visible spectrophotometry; Biosensors; Blood; Calibration; Genetic algorithm; Glucose; Multicriteria target vector optimization; Reagent strips

Multicriterial or polyoptimization means the simultaneous search of the desired values $\mathbf{y}(d) = [y_1(d), y_2(d), \ldots, y_m(d)]$ for a multivariate function f of the type

$$(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m) = f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k) \quad (1)$$

Correspondence to: D. Wienke, Laboratory for Analytical Chemistry, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands).

concerning m y variables (responses or criteria), which depend on k independent x variables.

One example of a multicriterial optimization problem in analytical chemistry is the optimization of $i = 1, \ldots, m$ distances between chromatographic peaks as a function f of $j = 1, \ldots, k$ chromatographic conditions (mobile phase composition, pH value, etc.) [1–12]. Another example is the simultaneous optimization of the intensities of $i = 1, \ldots, m$ emission lines in atomic spectrom-

etry of a mixture as function f of the $j = 1,\ldots,k$ excitation conditions for a given spectrometer type [13–24].

One method to approach the desired response vector $\mathbf{y}(d)$ is multicriterial target vector optimization [24], also known as goal programming [25] or the optative vector method [26,27]. Multicriterial target vector optimization means, in principle, the minimization of the vector distance $D = \mathbf{y}(d)-\mathbf{y}(a)$ of the (by the user) specified $\mathbf{y}(d)$ to the actually reached vector $\mathbf{y}(a)$ in the m-dimensional space of the response variables. This is done by searching in the k-dimensional x-space. Among other measures for D, the Euclidian distance is a possibility for describing the closeness between $\mathbf{y}(d)$ and $\mathbf{y}(a)$ [24–27], for example.

Thus the multicriterial target vector optimization allows the simultaneous minimization and/or maximization and/or the search for target values different from extremes of single variables within the total set of m responses via a maximization of the similarity of response vectors. In other words, multicriterial target vector optimization is a more general polyoptimization strategy in contrast to techniques that are dealing only with pure simultaneous maximization or with pure simultaneous minimization of all response variables. It should only be remarked that the correct weighting of the responses also determines the size of the distance D.

As stated in part I [24], not only the distance measure as a special optimization criterion but further the mathematical constraints, the means of obtaining the function f and the search technique used influence the success and the quality of a polyoptimization result. Especially the choice of the search technique is important for reaching the global or local optimum and if the k-dimensional search space can be checked representatively.

A newer search technique is the genetic algorithm (GA), developed by Holland [28,29] and recently also introduced into chemometrics. Kateman [30] placed the GA in general into the whole range of techniques using principles of natural evolution and parallel information processing such as learning expert systems and artificial neural networks (ANN).

Lucasius and co-workers [25–33] reported applications of the GA to optimize the fit between the structure of DNA molecules and their corresponding NMR spectra and also for wavelength selection problems in spectrometry. Bos and Weber [34] described the optimization of numerical parameters of an ANN with a GA and Fontain [35] discussed the use of GA for distance optimization of molecules in computer-aided synthesis design.

The use of the GA for multicriterial optimization of excitation parameters in atomic emission spectrometry was discussed in Part I [24]. It was found that the GA has a higher chance of reaching global optima at difficult structured response surfaces than classical search techniques.

The second important property of the GA, that of handling high-dimensional search spaces, will be discussed in this paper for the practical case of optimization of analytical procedures with m responses. High-dimensional search spaces exist when the number k of independent variables increases or/and if the resolution per variable increases. Then the search space size grows exponentially in k.

The target vector criterion was combined with a search technique based on a genetic algorithm for multicriterial optimization of the photometric calibration graph of a biochemical test strip for the determination of glucose in human blood. Attempts were made to optimize twelve chemical and technological parameters of the strip preparation to obtain simultaneously the desired sizes of six optical properties (responses) of the test strip.

This so-called indirect optimization via variation of the k independent variables was then compared with a direct multicriterial prediction on the basis of a classical and a non-linear partial least-squares (PLS) regression.

THEORY

The theory of multicriterial target vector optimization with different types of distance measures was discussed in detail in Part I [24]. The theory and practice of the genetic algorithm for

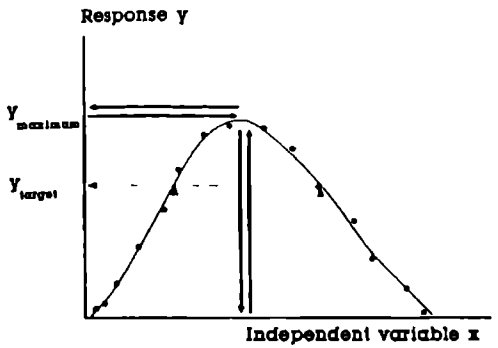

Fig. 1. A biunique solution for the (global) extremum (bold straight lines) For the target optimization problem (dotted lines) a non-biunique solution is obtained.

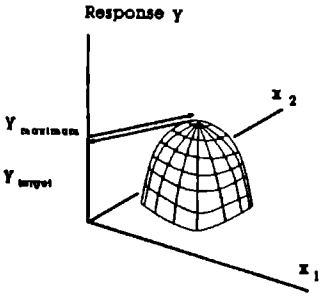

Fig. 2. Direct (deductive) optimization versus indirect (inductive) optimization of a multiple function $y = f(x_1, x_2, \ldots, x_k)$. As in Fig. 1 the deductive strategy yields a non-biunique solution again in comparison with the inductive method.

its use in multicriterial optimization was also outlined there. In this theoretical part, attention is focused more on the relationships, advantages and disadvantages of direct and indirect multicriterial optimization strategies.

From the univariate optimization problem in Fig. 1 it can be seen that for the exact (global) optimum no difference in the solutions between direct (deductive) and indirect (inductive) optimization strategy occurs (bold lines). According to the maximum, both projections behave biuniquely. However, if so-called target values are desired as optima other than from the extrema of the response function f, it becomes more difficult. Such a case is shown by the dotted lines in Fig. 1. A projection from x on to y via response function f yields in every case one solution. Sometimes it can be the same. A reverse projection from y on to x via f can yield for the given example two solutions. The situation is not longer biunique.

In the multiple case with $m = 1$ response variable y (Fig. 2) but $k > 1$ independent variables $(x_1, \ldots, x_k)$, a biunique projection for the global maximum can be found again. However, the number of possible target solutions can rise to infinity, obtained by the total number of all projections of the dotted curve on to the x_1–x_2 plane. The number of target solutions is commonly also larger for a univariate or a multiple case with more than one optimum. Even for the case of unlimited periodic functions such as sine or cosine functions with an unlimited number of global optima, the number of target solutions is usually larger than the number of extremes.

Further generalization to $m > 1$ responses (multicriterial case) yields new problems: the individual optima of the m response variables can differ from each other according to their position in the X space (Fig. 3). As in the multiple case (Fig. 2), all of them can be obtained by an individual biunique projection. For the multicriterial global compromise maximum y_{max} sometimes a biunique projection between the X and Y spaces can also exist. This depends on the overlayed

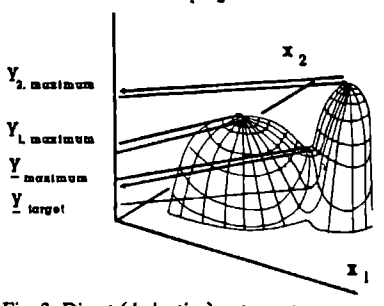

Fig. 3. Direct (deductive) optimization versus indirect (inductive) optimization of a multicriterial function $(y_1, y_2, \ldots, y_m) = f(x_1, x_2, \ldots, x_k)$. As in Figs. 1 and 2 the deductive strategy yields a non-biunique solution again in contrast to the indirect optimization, which gives a biunique (global) maximum $y_{maximum}$.

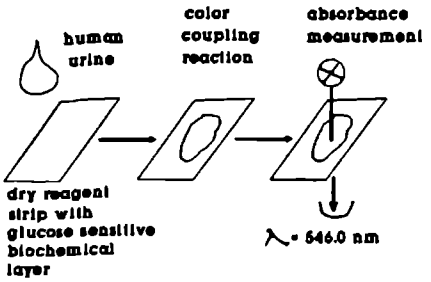

Fig 4 The principle of the multicriterial optimized dry reagent test strip for the determination of human glucose in clinical analysis (see text for details).

functions $y_1, y_2, \ldots, y_m$. In contrast to y_{max} for the multicriterial compromise target vector optimum y_{target}, in most instances no biunique projection exists (Fig. 3, projection of the dotted curves on to the x_1–x_2 plane).

Summarizing the theoretical considerations, it could be shown that the case of the target vector optimization problem is characterized much more often by a situation with disturbed biuniques between the two variables spaces than the case of the determination of classical optima, e.g., pure extrema. One way to overcome this problem is the so-called analytical solution. The extremes of a multicriterial function (Eqn. 1) can sometimes be found by looking for the roots of the first derivative of f. One can try to find desired target values different from the pure extremes by filling the desired values $\mathbf{y}(d)$ into f and trying to find the corresponding values $\mathbf{x}(d)$. Thus the analytical solution is the most straightforward deductive strategy. However, if there is no way to invert f the optimization problem can be solved by the inductive strategy: stepwise variation of the x variables using an efficient and save search technique, predicting $\mathbf{y}(a)$ for every actual search step $\mathbf{x}(a)$ on basis of f and minimizing $D = \mathbf{y}(d) - \mathbf{y}(a)$.

EXPERIMENTAL

As a chemical example, the optimization of the slope and the shape of the photometric calibration graph for a biochemical dry reagent test strip for glucose determination in human urine was chosen (Fig. 4). The test strip consists of a polymer layer coated with a thin gelatine film containing a glucose-sensitive combination of different biochemical reagents [36]. The concentrations of these reagents and the chemical process parame-

TABLE 1

Names and ranges of experimentally varied chemical and process parameters (x-variables) and corresponding measured properties of the glucose strip (y-variables)

Name of x-variable	Abbreviation	Experimentally varied range	Name of y-variable	Experimentally measured range
Chromium acetate solution (5%)	CrAc	1.5–3.0 ml [a]	$A5/4$ [c]	0.175–0.321
Glucose oxidase	GOD	522–20 880 units [a]	$A15/4$ [c]	0.330–0.670
Peroxidase	POD	378–9450 units [a]	$A25/4$ [c]	0.385–0.930
Developer	Dev	120–480 mg [a]	$A5/35$ [c]	0.155–0.250
Ethylene glycol	EthGly	0–2475 ml [a]	$A15/35$ [c]	0.320–0.560
pH of the casting medium	pH-C	6.7–8.0	$A25/35$ [c]	0.360–0.820
pH of the dry film surface	pH-F	6.94–7.85		
Water absorption after 1 min	H_2O	0.88–2.49 mg cm^{-1}		
Layer thickness	Thick	5.8–8.7 μm		
Support layer (1 = cellulose acetate, 2 = polyester)	Supp	1 or 2 [b]		
Conditions of drying (1 = slow, 2 = fast)	Dry	1 or 2 [b]		
Sodium sulphite	NaS	80–230 mg [a]		

[a] The figures refer to 100 ml of casting medium in each cast [b] 1 and 2 are used for processing nominal data. [c] $A15/4$ means the absorbance of 15 mmol l^{-1} glucose at 546 nm after 6 weeks of keeping at 4°C.

ters for the preparation of the strips form in total $k = 12$ independent variables (Table 1). Chosen combinations of certain levels of these variables determine the shape and the slope of the calibration graph for a strip in a theoretical unpredictable way. Therefore, it was decided to follow a systematic empirical strategy.

The calibration graph for a given strip was characterized by $m = 6$ y variables: the photometric absorbances for the three glucose concentration levels of 5, 15 and 25 mmol l^{-1} for a constant temperature at 4°C and after artificial ageing (6 weeks at 35°C) for the same three glucose levels again. The aim of the study was now to approach an optimum parameter set $\mathbf{x}(d)$ which yields via a function f according to Eqn. 1 a desired non-linear calibration graph $\mathbf{y}(d)$ for both the low and high temperature range with absorbance values of

$$\mathbf{y}(d) = \begin{pmatrix} y_1(d) \\ y_2(d) \\ y_3(d) \\ y_4(d) \\ y_5(d) \\ y_6(d) \end{pmatrix} = \begin{pmatrix} A5/4 \\ A15/4 \\ A25/4 \\ A5/35 \\ A15/35 \\ A25/35 \end{pmatrix} = \begin{pmatrix} 0.20 \\ 0.45 \\ 0.65 \\ 0.20 \\ 0.45 \\ 0.65 \end{pmatrix}$$

where $A1/2$ means the absorbance at 546.0 nm for a glucose concentration level 1 (mmol l^{-1}) for a constant temperature 2 (°C).

The preparation of the strips was carried out in two experimental steps: casting the films with variation of the presumably influencing parameters (Table 1) and cutting each film to a sufficient number of single strips for all investigations; and keeping two samples of the same film charge (represented by around 100 strips for one charge) for 6 weeks at two different temperatures (4 and 35°C), treatment of every charge with three glucose solutions (5, 15 and 25 mmol l^{-1}) and followed by the determination of the strip absorbance at 546 nm [ten parallel measurements with a Specol 20 spectrophotometer (Carl Zeiss, Jena, Germany)] giving the six y variables for every charge.

Keeping the samples for 6 weeks at 35°C simulates ageing for 1 year at room temperature. This was done because a second aim of the optimization was that the shape of the calibration graph should be independent of the keeping temperature.

By combination of classical factorial designs with other designs formed on the basis of results of a principal component analysis (so-called multivariate design [36–38]), in total 104 carefully selected experiments were carried out for simultaneous variation of the levels of the $k = 12$ independent variables within the limits given in Table 1. The experimentally varied ranges of the independent variables resulted from experience and from some earlier manual experiments. Further details can be found in [36,39]. These 104 experiments led to two mutually dependent matrices: the data matrix $\mathbf{X}_{104,12}$ of varied chemical and technological parameters and the data matrix $\mathbf{Y}_{104,6}$ of corresponding measured calibration graphs per dry reagent strip. These two data matrices formed the basis for the following multicriterial optimization studies.

All multivariate calculations (PLS, MLR) were done by the program package MULTIVAR [40,41] on an IBM-PC. Part of the multicriteria optimization was carried out by the program package POLYOPT [42] as an exhaustive grid search on an IBM-PC under MS-DOS. The other part was done with a program formed by the Genetic Algorithms Library Toolbox GATES [43,44] and the evaluation function $1/D$ used on a Sun-Sparc workstation under the UNIX operating system.

RESULTS AND DISCUSSION

Deductive polyoptimization with partial least-squares regression

The application of the classical PLS method as a deductive polyoptimization technique to the data sets $\mathbf{X}_{104,12}$ and $\mathbf{Y}_{104,6}$ was limited by potentially three critical points: (i) a larger number than $k = 12$ independent variables had to be predicted from a smaller number of $m = 6$ dependent variables; (ii) the relationship f between the two data matrices was expected to be non-linear; and (iii) this seeming multivariate non-linearity can potentially disturb the necessary biunique-

ness between the data sets as outlined in the theoretical section.

The observed difference between the optimum predicted by classical PLS [45] and the experimentally realizable optimum as reported recently [36] seemed to support the individual expectations (i)–(iii). However, a detailed analysis of the eigenvalues and the loading of the estimated classical PLS model showed that the limitations (ii) and (iii) are more important than (i). The square roots of the PLS eigenvalues of the autoscaled data sets are 22.96, 6.02, 5.62, 2 49, 1.77 and 1 09. Hence the importance of the extracted latent PLS variables decreases with a ratio of approximately 4 . 1 : 0.9 : 0.3 : 0.2 : 0.05.

The first and most important pair of latent variables influences all six absorbances of the test strip simultaneously (Fig 5). The other pairs of latent variables yield special corrections of the absorbances giving the characteristic non-linear shape and the temperature stability of the calibration graphs. This result, that all six y variables are completely controlled by the first three to four latent PLS variables (Fig. 5a), in principle, and that nearly the full variance of the data sets is stored in those first few latent variables allows two backward-oriented conclusions.

The first conclusion is that only those x variables are important for the desired calibration graph which contribute to the first four latent variables. The other remaining x variables have a minor influence on the strip quality. Second, the result means that obviously sets of x variables have a correlated impact on the six response variables (Fig. 5b), despite the fact that they were varied independently from each other using an experimental design

In the prediction step, such a set of x variables will be predicted together as a set of variables again. In other words, because of the special multivariate correlations within the data sets $X_{104,12}$ and $Y_{104,6}$, it can be concluded that the

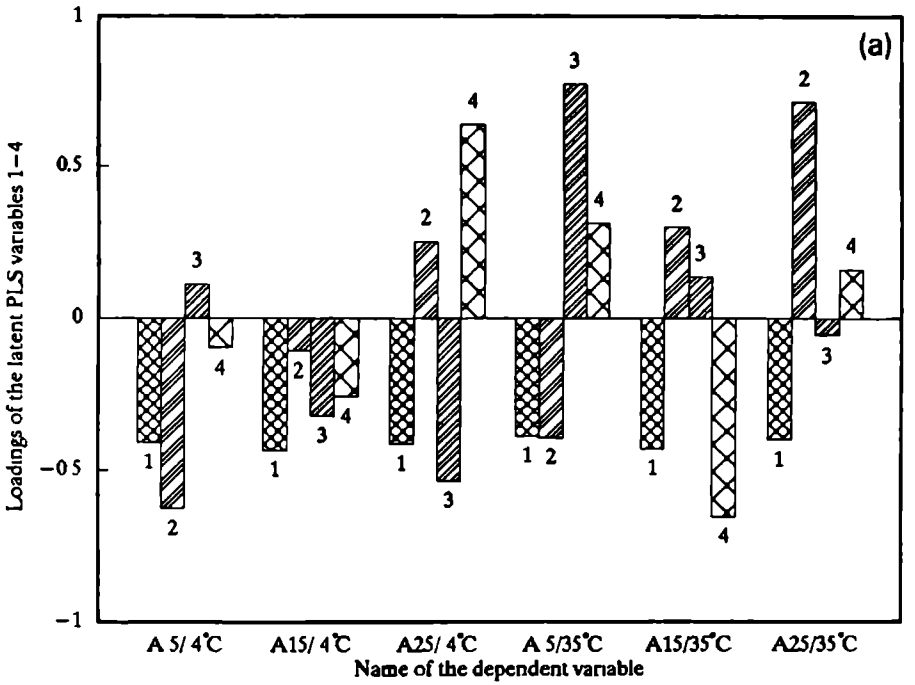

Fig 5 Loadings of the original (a) six *y*- and (b) twelve *x*-variables on the most important four latent variables extracted from the classical PLS model from the data sets $Y_{104,6}$ and $X_{104\,12}$.

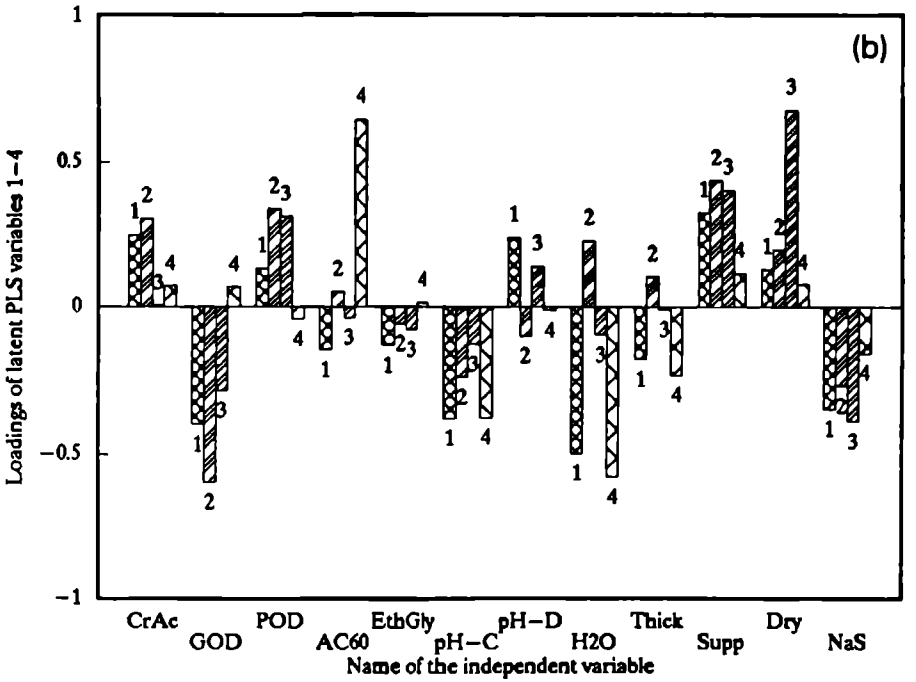

Fig. 5 (continued).

critical point (i), e.g., the prediction of twelve x variables on the basis of six y variables, can be ignored as a problem for those data sets. As an independent numerical security check for this hypothesis, a pseudo-non-linear PLS regression was performed. The data set $Y_{104,6}$ was extended by adding all fifteen binary interaction terms and the six pure squared terms giving in total a data matrix of $m = 27$ dependent variables. This number $m = 27$ now exceeded the number $k = 12$ of independent variables. Further, this introduction of additional second-order polynomial terms into the Y matrix with application of the classical PLS method (called PY-PLS) sometimes promises a more adequate fit to slightly non-linear data sets [46] attacking simultaneously with (i) also the second problem (ii) above.

A comparison of the x conditions used up to the optimization with the predicted optima (Table 2) on the basis of classical PLS, the realizable optimum, and on the basis of this PY-PLS, showed that the main improvement could be achieved by a drastic decrease in the concentration of the enzyme glucose oxidase (GOD). The recently given biochemical interpretation of this PLS results was that a useless excess of GOD causes early ageing of the strips and in this way a continuous change in the calibration graph for the strip [36,39]. On the other hand, the absolute concentration values of 505 units predicted by classical PLS and of −720 units predicted by PY-PLS for GOD were found intuitively to be too low, i.e., biochemically meaningless.

Thus 1044 units for the GOD concentration were realized as a practically more useful level. All other variables were set in the laboratory verification experiment and in the pilot plant as found by classical PLS (Table 2). In this way test strips with the desired long-term thermal stability and the expected non-linear calibration graphs could be synthesized as reported recently [36]. This means in practice that the polynomial MLR

model (Table 3, Figs. 6 and 7) is more able than the classical PLS to predict the correct levels for the twelve x variables to obtain a temperature-stable calibration graph. As a consequence, an advanced PLS model (PX-PLS) which uses the same modified X matrix as the MLR model was applied to the data sets. Additionally, a further pseudo-non-linear PLS version was implemented by replacing the linear inner relationship with a second-order polynomial [47,48].

The results of pseudo-non-linear PLS with data matrices extended by polynomial terms (PY-PLS, PX-PLS) and with a second-order polynomial for the inner relationship (PP-PLS) show the same tendencies (Table 2) in the predictions as classical PLS: minimize the initial GOD concentration and the ethylene concentrations, choose the fast drying mode DRY = 2 of the strip, use as the support layer SUPP = 2 (polyester material), etc. However, the absolute values predicted by the three non-linear PLS versions for GOD were found to be too low again and biochemically meaningless.

Summarizing the results of deductive polyoptimization with both classical and the pseudo-non-linear PLS models, it can be seen that the rank problem (i) can be solved and non-linearities (ii) can be treated by adapted PLS versions. The interpolation and the extrapolation of such non-linear models remains a dangerous step in general and should be avoided.

Additionally to this problem is the disturbed

TABLE 2

Comparison of the experimentally realized conditions for glucose test strip preparation before optimization with the predicted conditions after optimization by classical PLS, optimization by classical PLS with additional polynomial terms in the Y-matrix (PY-PLS) and in the X-matrix (PX-PLS) and polynomial PLS with second-order inner relationship (PP-PLS) [a]

	CrAc (ml)	GOD (units)	POD (units)	Dev (mg)	EthGly (ml)	pH-C	pH-D	H_2O (mg cm^{-1})	Thick (μm)	Supp	Dry	NaS (mg)
Typical example before optimization	3	10440	1890	240	0.75	7	7.3	1.3–1.6	6.4–7.0	1	2	120
Predicted with PLS	3 ±0.05	505 ±1141	4001 ±580	225 ±5	0.74 ±0.06	6.94 ±0.05	7.5 ±0.05	1.64 ±0.05	7.2 ±0.24	2 ±0.15	1.6 ±0.18	101 ±13
Verification experiments												
Lab charge R_1	3	1044	3800	225	0.75	7	7.51	1.52	8.0	2	2	100
Lab charge R_2	3	1044	3800	225	0.75	7	7.61	1.41	7.1	2	2	100
Lab charge R_3	3	1044	3800	260	0.75	7	7.5	1.34	6.8	2	2	100
Lab charge R_4	3	1044	3800	225	0.75	7	7.5	1.58	7.8	2	2	100
Lab charge R_5	3	1044	3500	225	0.75	7	7.66	1.46	7.5	2	2	100
Pilot plant control expt.	3	1566	3800	225	0.75	7	7.5	1.70	8.5	2	2	100
Predicted with PY-PLS	3.1 ±0.05	−720 ±1046	4425 ±530	248 ±15	0.78 ±0.07	6.9 ±0.04	7.5 ±0.05	1.56 ±0.04	7.2 ±0.23	2.2 ±0.14	2 ±0.16	92 ±11
Predicted with PX-PLS	3 ±0.05	59 ±1170	4030 ±585	222 ±15	0.68 ±0.06	7 ±0.05	7.5 ±0.06	1.66 ±0.04	7.3 ±0.24	2 ±0.15	1.7 ±0.19	106 ±13
Predicted with PP-PLS [b]	3 ±0.05	333 ±1100	3900 ±490	220 ±15	0.75 ±0.06	6.87 ±0.05	7.5 ±0.05	1.65 ±0.05	7.2 ±0.23	2 ±0.14	1.9 ±0.17	100 ±0.13

[a] The conditions of the verification experiments done after the PLS optimization are also given. The confidence intervals of prediction (±) were approximated by cross-validation. [b] Original data sets without any added polynomial terms.

TABLE 3

Polynomial multivariate regression model for the 104 experimental data points, six dependent and twelve independent variables and two squared and four binary interaction terms

y	Intercept	CrAc pH-F GOD · GOD	GOD H_2O H_2O · Thick	POD Thick pH-C · H_2O	Dev Supp GOD · POD	EthGly Dry Dev · CrAc	pH-C NaS POD · POD
A5/4	$2.94552 \cdot 10^{-1}$	$-6.7388 \cdot 10^{-3}$ $1.10951 \cdot 10^{-2}$ $4.39602 \cdot 10^{-3}$ $-2.2682 \cdot 10^{-2}$	$6.42062 \cdot 10^{-2}$ $3.08577 \cdot 10^{-3}$ $-2.4210 \cdot 10^{-2}$	$3.83755 \cdot 10^{-2}$ $1.68062 \cdot 10^{-2}$ $2.43028 \cdot 10^{-2}$	$1.24733 \cdot 10^{-2}$ $1.79498 \cdot 10^{-2}$ $-6.3282 \cdot 10^{-3}$	$1.06748 \cdot 10^{-3}$ $-8.0684 \cdot 10^{-3}$ $4.81782 \cdot 10^{-2}$	 $6.49459 \cdot 10^{-3}$ $-3.1011 \cdot 10^{-3}$
A15/4	$5.87667 \cdot 10^{-1}$	$-6.6624 \cdot 10^{-4}$ $2.12030 \cdot 10^{-2}$ $-6.9812 \cdot 10^{-3}$ $-5.1551 \cdot 10^{-2}$	$2.94308 \cdot 10^{-2}$ $-6.2058 \cdot 10^{-3}$ $-5.1977 \cdot 10^{-2}$	$-5.1930 \cdot 10^{-3}$ $8.37106 \cdot 10^{-2}$ $5.58876 \cdot 10^{-2}$	$3.72161 \cdot 10^{-2}$ $3.07169 \cdot 10^{-2}$ $-6.1353 \cdot 10^{-2}$	$9.46608 \cdot 10^{-3}$ $-7.6402 \cdot 10^{-3}$ $5.16195 \cdot 10^{-4}$	 $-1.8793 \cdot 10^{-3}$ $3.15741 \cdot 10^{-3}$
A25/4	$7.92998 \cdot 10^{-1}$	$1.22195 \cdot 10^{-2}$ $6.10983 \cdot 10^{-3}$ $-1.1307 \cdot 10^{-2}$ $-1.1525 \cdot 10^{-1}$	$-7.3914 \cdot 10^{-2}$ $-7.1871 \cdot 10^{-3}$ $-1.0268 \cdot 10^{-1}$	$-1.2222 \cdot 10^{-1}$ $1.16627 \cdot 10^{-1}$ $1.04064 \cdot 10^{-1}$	$1.07343 \cdot 10^{-1}$ $3.92732 \cdot 10^{-2}$ $-1.6823 \cdot 10^{-1}$	$1.22244 \cdot 10^{-2}$ $-6.6063 \cdot 10^{-4}$ $-1.2720 \cdot 10^{-1}$	 $-1.2799 \cdot 10^{-2}$ $1.57017 \cdot 10^{-2}$
A5/35	$2.57727 \cdot 10^{-1}$	$-6.5504 \cdot 10^{-3}$ $1.88586 \cdot 10^{-3}$ $5.46664 \cdot 10^{-4}$ $-2.4567 \cdot 10^{-2}$	$3.84279 \cdot 10^{-2}$ $-1.2294 \cdot 10^{-3}$ $-2.9051 \cdot 10^{-2}$	$2.79137 \cdot 10^{-2}$ $1.45249 \cdot 10^{-2}$ $1.77412 \cdot 10^{-2}$	$1.89933 \cdot 10^{-2}$ $1.23935 \cdot 10^{-2}$ $2.53629 \cdot 10^{-3}$	$2.53392 \cdot 10^{-3}$ $-3.3727 \cdot 10^{-3}$ $3.15507 \cdot 10^{-2}$	 $7.33487 \cdot 10^{-3}$ $1.84913 \cdot 10^{-3}$
A15/35	$4.97370 \cdot 10^{-1}$	$-4.7199 \cdot 10^{-3}$ $-1.0128 \cdot 10^{-2}$ $-3.4816 \cdot 10^{-4}$ $-5.4247 \cdot 10^{-2}$	$3.72093 \cdot 10^{-3}$ $-1.0675 \cdot 10^{-3}$ $-5.8509 \cdot 10^{-2}$	$-3.2711 \cdot 10^{-3}$ $9.83987 \cdot 10^{-2}$ $4.94254 \cdot 10^{-2}$	$2.57522 \cdot 10^{-2}$ $2.24904 \cdot 10^{-2}$ $-4.1943 \cdot 10^{-2}$	$2.18397 \cdot 10^{-3}$ $1.76268 \cdot 10^{-3}$ $-5.7852 \cdot 10^{-3}$	 $-1.3629 \cdot 10^{-3}$ $-2.4250 \cdot 10^{-3}$
A25/35	$6.95756 \cdot 10^{-1}$	$2.89603 \cdot 10^{-3}$ $-3.7112 \cdot 10^{-2}$ $-1.1005 \cdot 10^{-3}$ $-1.2553 \cdot 10^{-1}$	$-7.8908 \cdot 10^{-2}$ $-1.3917 \cdot 10^{-3}$ $-1.0694 \cdot 10^{-1}$	$-8.2873 \cdot 10^{-2}$ $1.72733 \cdot 10^{-1}$ $1.08731 \cdot 10^{-1}$	$9.47050 \cdot 10^{-2}$ $3.90166 \cdot 10^{-2}$ $-6.8523 \cdot 10^{-2}$	$-6.8387 \cdot 10^{-3}$ $5.06819 \cdot 10^{-3}$ $-9.1130 \cdot 10^{-2}$	 $-6.0565 \cdot 10^{-3}$ $2.09326 \cdot 10^{-2}$

biuniqueness, which cannot simply be rejected by some modifications of the existing PLS theory. The only way to overcome this problem [(iii); see p. 257] and to overcome the danger of an arbitrary extrapolation is a reverse strategy, i.e., inductive polyoptimization.

TABLE 4

Results of multicriterial target vector optimization with grid search using variables resolution RES = 3 and RES = 5 between the runs but constant variables resolution within the runs [a]

RES	Grid size	$1/D$	CrAc (ml)	GOD (units)	POD (units)	Dev (mg)	EthGly (ml)	pH-C	pH-D	H_2O (mg cm^{-1})	Thick (μm)	Supp	Dry	NaS (mg)
3	531 441	50.0	3.0	522	4914	120	0.00	6.70	7.40	1.685	8.70	2.0	2.0	155.25
5	244 140 625	89.5	3.0	5611	4914	120	0.00	6.70	7.16	1.685	5.80	2.0	1.0	80.50

	Corresponding predicted responses				
	A5/4	*A15/4*	*A25/4*	*A5/35*	*A15/35*	*A25/35*
3	0.2020	0.4531	0.6532	0.1844	0.4404	0.6541
5	0.2057	0.4555	0.6472	0.1943	0.4462	0.6475

[a] $1/D$ = reciprocal Euclidian distance of that best member of the whole search population that is the closest to the desired target vector $y(d)$.

Inductive polyoptimization with grid search and genetic algorithm

The verified MLR model (Table 3, Figs. 6a,b and 7a,b) was introduced into the grid search module of POLYOPT [38] together with the inverted Euclidian distance $1/D$ as measure of fitness. Because of the significant size of the higher order terms in the MLR model and their different signs, a response surface with numerous local optima was expected.

Grid searching as an exhaustive technique was applied to obtain a raw overview of the twelve-dimensional hypersurface with the advantage of acquiring all local and the global polyoptima corresponding a chosen resolution, RES.

The only, but major, disadvantage of exhaustive searching is the exploding size of the search space size S:

$$S = \prod_{j=1}^{k} (\mathrm{RES}), \qquad (2)$$

as a product of the single resolutions of the k x variables. The grid sizes for RES = 3 and RES = 5 (Table 4) yield a large number of search positions. By evaluation of 500 grid positions per minute with POLYOPT on an IBM-PC (Intel 80486, 33 MHz), the case for RES = 5 required 34 days of computing time. This experiment was done also to control the results of GATES.

For GATES the same MLR model (Table 3) was used and also the inverted Euclidian distance as measure of fitness. Further parameter settings of the GA program are given in Table 5.

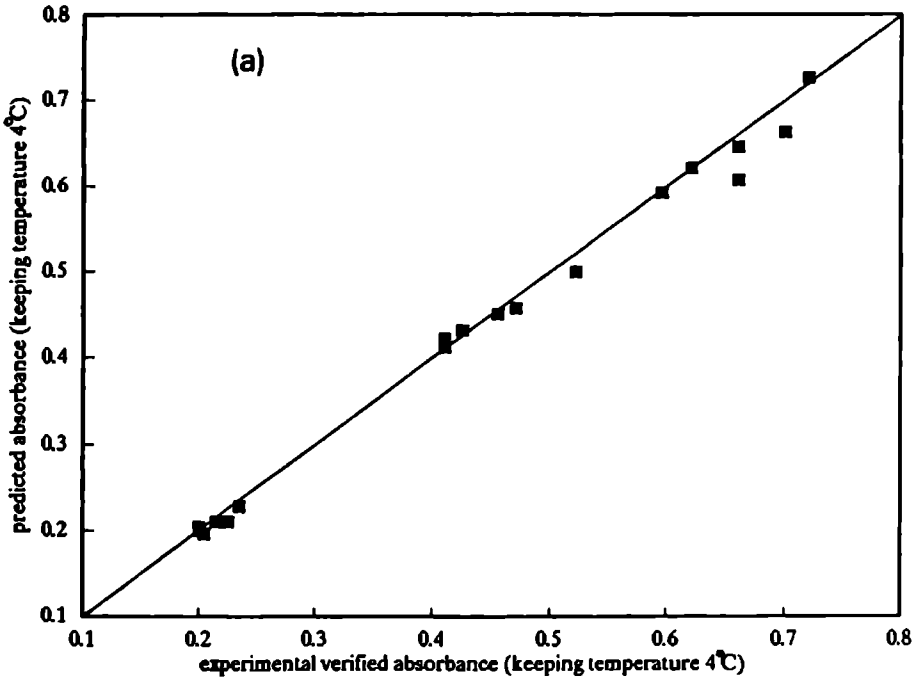

Fig. 6. Independent validation of the predictive power of the developed multivariate polynomial regression model (MLR) by plotting the results for all six response variables simultaneously obtained by five additional control experiments in a laboratory- and one large-scale experiment in a pilot plant (see also Table 4) versus the values for the six response variables predicted by the MLR. The calibration of the model was carried out with 104 designed experiments. Keeping temperature: (a) 4°C, (b) 35°C.

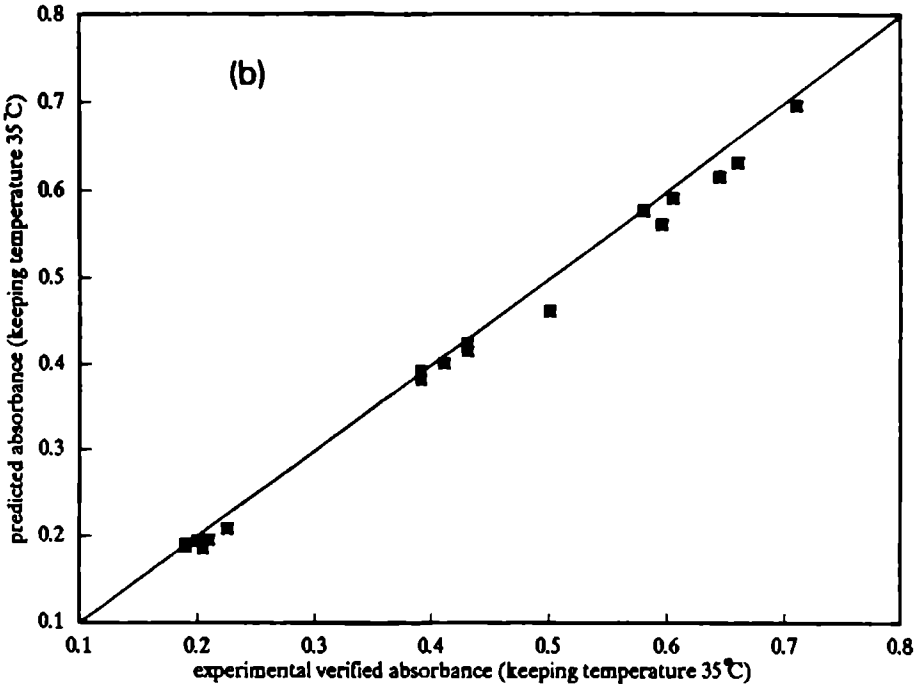

Fig. 6 (continued).

The computation time for the corresponding GA run for RES = 5 at a SunSparc Workstation was a few minutes of real time. A transformation of this time corresponds to approximately 1 h on the personal computer used for POLYOPT. After this hardware correction the gain in computing time by the use of the genetic algorithm in contrast to the grid search was at least 1:800 for RES = 5. This gain in computing time increases exponentially with a linear increase in RES.

For the resolution RES = 5 the same global optimum was detected by GATES as by an exhaustive search with POLYOPT (Tables 4 and 6). A useful graphical expression for the convergence control of the GA is the running measure of fitness $1/D$ (Fig. 8).

The polyoptimum obtained for the poor resolution of RES = 3 (Table 4) already shows a close similarity to the results obtained with the PLS versions (Table 2) using the experimental ranges per variable as a basis for a comparison (Table 1). Going now from resolution RES = 3 to a higher value RES = 5 (Table 4), larger changes only for the variables GOD, layer thickness, drying conditions and sodium sulphate can be observed. The low variable resolutions of 3 and 5 are not valid yet for a fair competition with the PLS techniques. PLS, in principle, samples the X space continuously. Therefore, the discrete resolution of the search space for the GA was increased up to 7 bits, e.g., 128 + 1 levels per variable. Such an increase allows nearly continuous sampling of the X space during the search.

On the other hand, this increase in resolution potentially also can increase the number of dangerous local optima rapidly. To overcome this problem partially, all runs of the GA presented in Table 6 were repeated five times independently with different initial parameter settings. Thus the predicted optima (Table 6) came close to the PLS predictions, in general, but the initial GOD concentrations were now predicted also with a low

tendency, i e., 3728 units, but biochemically much more meaningful than with PLS.

However, as shown experimentally in Part I [20], the GA provides no guarantee of finding the global optimum, but potentially it has a higher chance of reaching global optima than the search techniques known up to now. In addition to the repetition of a run, in practice the possibility exists of choosing an individually adapted search step width per single variable based on physically and chemically reasonable interval sizes and based on the expected individual experimental error of the considered x variable. In this way a drastic reduction of the search space takes place but keeping the important information. The global optima can be reached more easily. The results of such an experiment confirm this hypothesis with rapidly improved measures of fitness (Table 7) All predicted optimum levels of the variables also came closer to the PLS predictions but the GOD remained at a meaningful level of around 3600 units.

It should further be remarked that all inductive optimization runs with POLYOPT and GATES were performed within the experimentally given individual variables ranges (Table 1), which means an interpolation only during the search via the MLR model (Table 3) as a less dangerous mathematical operation in contrast to an arbitrary extrapolation Such an extrapolation sometimes happens by application of a deductive strategy such as PLS if this is necessary to reach the desired polyoptimum.

Conclusions

Multicriterial target optimization can in principle be carried out in two ways First, one can try to project directly a desired vector of the re-

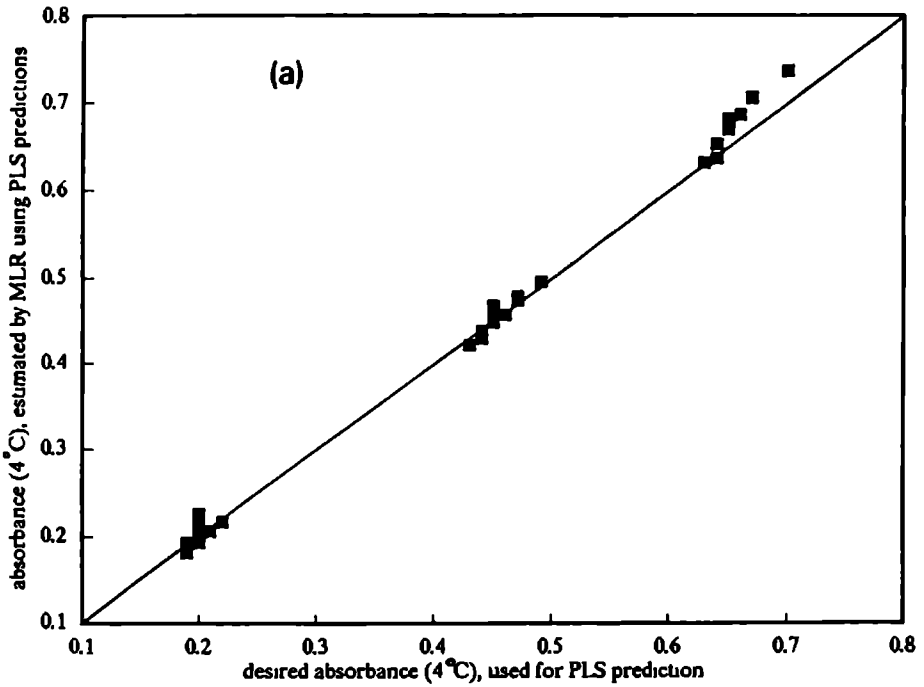

Fig 7 Comparison of the predictive power of the classical PLS model with that of the multivariate polynomial regression model MLR) for all six response variables simultaneously for five additional control experiments in a laboratory- and one large-scale experiment in a pilot plant (see also Table 4) The calibration of both models was carried out with 104 designed experiments Temperature (a) 4°C, (b) 35°C

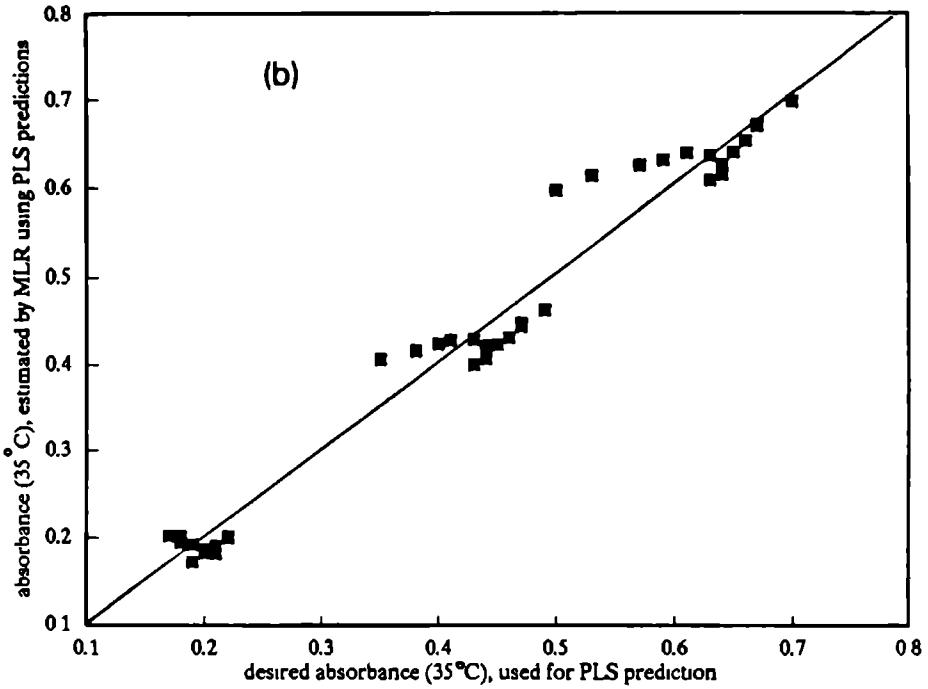

Fig 7 (continued)

TABLE 5

Set-up parameters and their practical meaning for the genetic algorithm toolbox GATES used for the multicriterial optimization of the glucose data [a]

Algorithmic variable	Setting	Practical meaning
No of bitstrings	400	Size of the search population
Bitstring format	[0–9,*], k RES	k bitstrings with resolution RES, 0–9 bits
Level count	128	Digitization of every variable
Gray coding	Yes	Others binary, natural coding
Max number of generations	1500	Optimization search steps
Fitness blow up	3 0	Dynamic evaluation of fitness quality
Enforced relative best fitness	9 0	
Elitist count	20	Control of selective reproduction
Mutation probability (start/end/rate)	0 01/0 01/0 0	During all generations constant 1% of the strings mutate
Cross swap	Segmented	How the strings exchange their information, alternative uniform
Cross swap probability (start/end/rate)	0 10/0 10/0 00	Bits are swapped if $P > 1\%$ and with constant probability
Cross mate probability (start/end/rate)	0 95/0 95/0 00	
Sharing feel scope	No share	Population may not split into subpopulations during search
Sharing feel blow-up	2 0	No meaning if no share allowed
Sharing probability	0 5/0 5/0 0	No meaning if no share allowed

[a] For further theoretical details see also [28,29].

TABLE 6

Results of multicriterial target vector optimization with the genetic algorithm using increasing variables resolution RES between the runs but constant variables resolution within the runs [a]

RES	1/D	Generations	CrAc (ml)	GOD (units)	POD (units)	Dev (mg)	EthGly (ml)	pH-C	pH-D	H_2O (mg cm^{-1})	Thick (μm)	Supp	Dry	NaS (mg)
5	89.5	129	3.0	5611	4914	120	0.0	6.7	7.16	1.68	5.8	2.00	1.00	80.5
9	90.4	151	3.0	5611	4914	120	0.0	6.7	7.05	1.68	5.8	2.00	1.00	80.5
17	90.1	707	2.6	4339	5481	120	0.3	6.7	7.73	1.78	6.0	1.93	1.56	80.5
129	107.8	1197	2.5	3728	5236	134	0.0	6.7	7.22	1.79	5.8	1.91	1.91	81.6

	Corresponding predicted responses					
	A5/4	*A15/4*	*A25/4*	*A5/35*	*A15/35*	*A25/35*
5	0.2057	0.4555	0.6472	0.1943	0.4462	0.6475
9	0.2050	0.4571	0.6490	0.1946	0.4465	0.6478
17	0.2077	0.4558	0.6494	0.1946	0.4493	0.6499
129	0.2003	0.4548	0.6509	0.1950	0.4442	0.6483

[a] $1/D$ = reciprocal Euclidian distance of that best member of the whole search population that is the closest to the desired target vector $y(d)$.

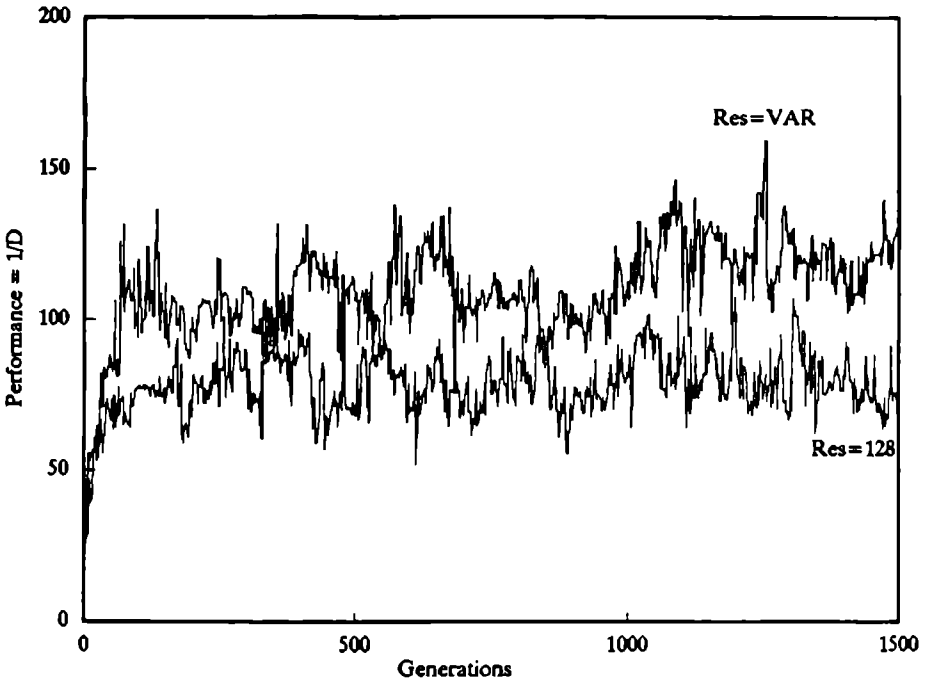

Fig 8. The inverse Euclidian distance $1/D$ as measure of fitness between the desired response vector $y(d)$ and the best response vector $y(a)$ really reached during the search for an actual search position (a) in the twelve-dimensional x-variables space using the genetic algorithm.

sponse variables via a response surface model function f on to the space of independent variables. This is called direct or deductive polyoptimization and can be performed in practice with, for example, an inverse multivariate regression technique such as partial least-squares regression. The other possibility is the search in the **X** space, predicting the $\mathbf{y}(a)$ vector for the actual search position and comparing it with a desired response vector $\mathbf{y}(d)$, for example, on the basis of a vector distance measure. This search can be carried out by multicriteria target vector optimization with a genetic algorithm as an efficient search method.

The advantage of the first strategy is that it works straightforwardly and that it requires one step for the estimation of the model and one step for the prediction of the optimum vector $\mathbf{x}(o)$ as an approximation of the desired $\mathbf{x}(d)$. A disadvantage of this deductive strategy is that it can only be applied if a biunique relationship between **Y** and **X** space exists. Another disadvantage is the potential danger of an arbitrary, difficult to control and not allowed extrapolation of the **X** space to reach the desired polyoptimum.

If the necessary biuniqueness is disturbed, the inductive strategy helps to find $\mathbf{x}(o)$. A search in the **X** space can always be performed. A further advantage is that interpolation and extrapolation of the **X** space can be strictly controlled.

Disadvantages of the inductive polyoptimization strategy are that it requires, like the deductive strategy, only one step also for the estimation of the response model function but numerous repetitions of the prediction step during the optimization search for the comparison of $\mathbf{y}(d)$ and $\mathbf{y}(a)$. Hence the inductive strategy is more time consuming in general. This happens especially if the number of search positions rises with an increase in the number of variables and/or an increase in the resolution of the variables. This disadvantage can be partly compensated by the GA as a parallel and evolutionary search technique, as was shown for the described optimization problem of dry reagent strips. The second problem for the indirect polyoptimization strategy is undesirable local optima. For this situation the GA also offers a practical solution.

Summarizing the results in Part I and this paper, it can be concluded that multicriteria target vector optimization on the basis of a suitable

TABLE 7

Results of five independent repeated runs of the multicriterial target vector optimization with the genetic algorithm using individual variables resolution and different numbers of generations [a]

Run	$1/D$	Generations	Individual resolution (step width)											
			128	128	64	64	32	128	64	128	64	4	4	64
			CrAc (ml)	GOD (units)	POD (units)	Dev (mg)	EthGly (ml)	pH-C	pH-D	H_2O (mg cm^{-1})	Thick (μm)	Supp	Dry	NaS (mg)
1	138.3	571	2.5	3565	5993	154	0.0	6.7	7.44	1.80	5.9	2.00	2.00	85.2
2	159.3	1250	2.9	3728	5562	143	0.1	6.7	7.20	1.82	5.8	2.00	2.00	84.0
3	145.6	998	2.8	3565	5993	149	0.0	6.7	7.30	1.86	5.8	2.00	2.00	81.7
4	154.9	209	3.0	3565	6424	160	0.0	6.7	7.18	1.86	5.8	2.00	2.00	84.0
5	154.1	1204	2.7	3565	6139	160	0.0	6.7	7.37	1.84	5.8	2.00	2.00	87.5

	Corresponding predicted responses					
	A5/4	*A15/4*	*A25/4*	*A5/35*	*A15/35*	*A25/35*
1	0.2037	0.4537	0.6496	0.1965	0.4464	0.6504
2	0.2032	0.4531	0.6486	0.1967	0.4475	0.6512
3	0.2025	0.4536	0.6488	0.1953	0.4480	0.6500
4	0.2024	0.4546	0.6495	0.1963	0.4492	0.6504
5	0.2026	0.4528	0.6502	0.1953	0.4478	0.6505

[a] The individual variables resolutions were chosen with respect to their importance and impact on the y-variables seen from the size of the PLS loadings. $1/D$ = reciprocal Euclidian distance of that best member of the whole search population that is the closest to the desired target vector $\mathbf{y}(d)$.

similarity measure between desired and actual response vectors and combined with a GA search technique is a powerful tool for optimizing simultaneously a set of responses of an analytical procedure or of a chemical product as discussed on the examples from atomic emission spectrometry and of the photometric calibration graph for a glucose test strip for clinical analysis.

REFERENCES

1 S.L. Morgan and S.N. Deming, J. Chromatogr., 112 (1975) 267.
2 R.J. Laub and J.H. Purnell, J. Chromatogr., 112 (1975) 71.
3 B. Sachok, R.C. Kong and S.N. Deming, J. Chromatogr., 199 (1980) 317.
4 J.L. Glajch, J.J. Kirkland, K.M. Squire and J.M. Minor, J. Chromatogr., 199 (1998) 57.
5 C.E. Goewie, J. Liq. Chromatogr., 9 (1986) 1431.
6 P.J. Schoenmakers, J. Liq. Chromatogr., 10 (1987) 1865.
7 D.L. Massart, B.G.M. Vandeginste, S.N. Deming, Y. Michotte and L. Kaufman, Chemometrics—a Textbook Elsevier, Amsterdam, 1988.
8 P.J. Schoenmakers, Optimization of Chromatography Selectivity, Elsevier, Amsterdam, 1986.
9 W. Wittkowski and J. Luthe, Chem. Technol., 36 (1984) 117.
10 S.N. Deming, J. Chromatogr., 550 (1991) 15.
11 H.R. Keller, D.L. Massart and J.P. Brans, Chemometr. Intell. Lab. Syst., 11 (1991) 175.
12 B. Bourguignon and D.L. Massart, J. Chromatogr., 586 (1991) 11.
13 G.L. Moore, P.J. Humphries-Cuff and A.E. Watson, Spectrochim. Acta, Part B, 39 (1984) 915.
14 H.A. Spaink, T.T. Lub, G. Kateman and H.C. Smit, Anal. Chim. Acta, 184 (1986) 87.
15 R.M. Belchamber, D. Betteridge, A.P. Wade, A.J. Cruickshank and P. Davison, Spectrochim. Acta, Part B, 41 (1986) 503.
16 G.S. Pyen, S. Long and R.F. Browner, Appl. Spectrosc., 40 (1986) 264.
17 M.S. Hendrick and R.G. Michel, Anal. Chim. Acta, 192 (1987) 183.
18 L. Ebdon, P. Norman and S.T. Sparkes, Spectrochim. Acta, Part B, 426 (1987) 619.
19 P. Werner and H. Friege, Appl. Spectrosc., 41 (1987) 32.
20 O.A. Guell and J.A. Holcombe, Spectrochim. Acta, Part B, 43 (1987) 459.
21 B. Naumann, B. Knull, F. Kerstan and J. Opfermann, J. Anal. At. Spectrom., 3 (1988) 1121.
22 S. Greenfield, M.S. Salman, M. Thomson and J. Tyson, J. Anal. Atom. Spectrom., 4 (1989) 55.
23 D. Wienke and K. Danzer, Fresenenius' Z. Anal. Chem., 342 (1992) 1.
24 D. Wienke, C. Lucasius and G. Kateman, Anal. Chim. Acta, 265 (1992) 211.
25 L.E.S. Khuri and F.U. Conion, Technometrics, 23 (1981) 363.
26 D. Wienke and K. Danzer, presented at Euroanalysis VII/COBAC V, Vienna, 1990.
27 D. Wienke, in J. Havel and M. Meloun (Eds.), Proc. 2nd Czechoslovakian Conference on Chemometrics, University of Brno, 1990, p. 34.
28 J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
29 D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
30 G. Kateman, Analyst, 115 (1990) 487.
31 C.B. Lucasius and G. Kateman, in J.D. Schaffer (Ed.), Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1989, p. 170.
32 C.B. Lucasius, M.J.J. Blommers, L.M.C. Buydens and G. Kateman, in L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, p. 251.
33 C.B. Lucasius and G. Kateman, Trends Anal. Chem., 10 (1991) 254.
34 M. Bos and H.T. Weber, Anal. Chim. Acta, 247 (1991) 97.
35 E. Fontain, Anal. Chim. Acta, 265 (1992) 226.
36 M. Ehrlich, D. Wienke, K. Danzer and G. Kateman, Fresenius' J. Anal. Chem., 343 (1992) 532.
37 S. Wold, M. Sjostrom, R. Carlson, T. Lundstedt, S. Hellberg, B. Skargerberg, W. Wikstrom and J. Ohmann, Anal. Chim. Acta, 191 (1986) 17.
38 D. Wienke, K. Danzer, M. Gitter, J. Aures, U. Munch, H.G. Byhan and H.J. Pohl, Anal. Chim. Acta, 223 (1989) 247.
39 M. Ehrlich, PhD Thesis, University of Jena, in preparation.
40 D. Wienke, MULTIVAR, Software Package for Multivariate Data Analysis, User Documentation, University of Jena, Jena, 1990.
41 D. Wienke, PhD Thesis, University of Jena, 1990.
42 D. Wienke, POLYOPT—Program Multicriteria Target Vector Optimization, User Documentation, 1991 unpublished.
43 C.B. Lucasius and G. Kateman, Comput. Chem., submitted for publication.
44 C.B. Lucasius, PhD Thesis, University of Nijmegen, in preparation.
45 M.A. Sharaf, D.L. Illman and B.R. Kowalski, Chemometrics, Wiley, New York, 1986.
46 S. Clementi, G. Cruciani, G. Curti and B. Skagerberg, J. Chemometr., 2 (1989) 499.
47 S. Wold, N. Kettaneh-Wold and B. Skagerberg, Chemometr. Intell. Lab. Syst., 7 (1989) 53
48 V.M. Taavitsainen and P. Korhonen, Chemometr. Intell. Lab. Syst., 14 (1992) 185.

CHAPTER 9

A New Approach to Curve Fitting Using Natural Computation

Contents

A New Approach to Curve Fitting Using Natural Computation

Abstract

In curve fitting, the most commonly used type of search technique is iterative hill-climbing that makes use of partial derivatives to proceed along the steepest path to an optimum in the solution space. A drawback of such a "greedy", or steepest descent, approach is that in order to find acceptable solutions, reliable and accurate initial estimates of the number of peaks, individual peak positions, heights, and widths are necessary – especially as the extent of peak overlap increases, which makes the problem more ill-conditioned. A consequence of this is that small errors in the data (e.g. noise or baseline distortions), errors in the mathematical model, or errors in the estimates, can be amplified, leading to large errors in the values obtained for the parameters of the final model. Moreover, the presence of overlapping peaks may lead to ambiguous fitting results – a general problem in curve fitting, i.e. is not limited to steepest descent methods only. In this article, we present a method for peak detection using an artificial neural network. Its result is used as a starting point for a genetic algorithm to perform the curve fitting. An important advantage of a genetic algorithm over the steepest descent approach in general, is the global search behavior: accurate initial estimates are no longer needed, as the chances to end up in local optima are considerably less. These statements are corroborated in our comparative case study, which involves the fitting of a series of spectra with strongly overlapping peaks: X-ray equator diffractometer scans of poly(ethylene) naphthalate yarns.

1 Introduction

Curve fitting can be a long and complicated task when the mathematical model underlying the peak pattern is not known, or when proper estimates for the parameters in the model assumed cannot be obtained. Moreover, a good fit of an experimental spectrum does not always lead to parameter values with a valid physical meaning, because a high degree of overlap between peaks may lead to ambiguities: different sets of parameter values give a close fit of the profile.

Vandeginste and De Galan evaluated curve fitting in infrared spectrometry [23]. They investigated the influence of the degree of overlap, the number of unresolved bands in the profile, and the determination of the baseline position, using both theoretical and experimental spectra. Furthermore, they formulated conditions which are to be fulfilled in order to obtain reliable results from the fit of infrared data.

J.A. Pierce *et al.* formulated the main sources of error in curve fitting [17]:

1. The exact number of peaks is not known, leading to poor convergence or to ambiguous solutions;
2. Uncertainty about the baseline position;
3. The initial estimates for the parameters in the model are insufficiently accurate; this may cause the problem that the fitting procedure ends up in a local optimum.

Error sources 1 and 2 originate from the experimental data, including the way the latter are collected, and prepared for the optimization method used subsequently for curve fitting; these error sources are thus independent of the optimization method. In contrast, error source 3 originates from the optimization method.

Traditionally, locally searching optimization methods are most widely applied, e.g. the Gauss-Newton minimization method for non-linear models, and other steepest descent methods. In contrast, globally searching optimization methods are more robust, i.e. less sensitive to initial estimates. Admittedly, globally searching optimization methods are computationally more intensive as well, but with the advent of increasingly faster and cheaper computers, computation times within practical time limits have become feasible.

This article presents a novel computational methodology gleaned from nature – *viz.* an artificial neural network, or ANN, combined sequentially with an artificial evolution procedure, or genetic algorithm, or GA – to tackle curve fitting

problems with the intent to minimize errors of the above three types.

2 The need for peak detection methods

In preparation for curve fitting, one needs to detect bands, i.e. peaks, in the data set first. A peak detection method is said to perform well if it provides an adequate estimate of the positions of the peaks (which, of course, implies their number); that is, if the error of the first kind (error source 1) is small.

For peak detection methods that use derivatives, it is important that the SNR (signal-to-noise ratio) in the original spectrum or chromatogram is high [17, 9]. A popular, advanced method to improve the SNR is due to Savitzky and Golay [20].

Jackson and Griffiths examined Fourier self-deconvolution (FSD), which comprises a linear method of deconvolution based on measured data in the time domain [9]. This technique seems to perform better than techniques that use derivatives, but it needs estimates of the peak shapes and "halfwidths" [10]. (For the sake of brevity, we use the term "halfwidth" to denote "full width at half maximum height" hereafter.)

Recently, a CMAC (cerebellar model arithmetic computer ANN) was developed for deconvolution of a system of two overlapping chromatographic peaks. A CMAC is able to provide rapid deconvolutions of both simulated and real chromatographic peaks [4]. An alternative ANN for peak detection is presented in the next subsection.

2.1 Peak detection using artificial neural networks

Our alternative ANN for peak detection, estimates the number of peaks, and their positions and halfwidths, in spectra, diffractograms, or chromatograms, that are composed of bell-shaped peaks. Since recently, ANNs are frequently applied with great success in areas such as non-linear calibration and pattern recognition [26, 22, 25]. ANNs differ from many other approaches in that they learn from examples. The ANN algorithm iteratively samples the examples and learns from these the mistakes made in previous trials. This process continues until all examples are mapped in a satisfactory way. For further reading, we recommend [19].

The ANN discussed here was trained to estimate the number of bands, and their positions and widths, in experimental spectra. A series of synthetic patterns of peaks varying from Lorenzian to Gaussian shapes with different degrees of overlap were used to train the ANN. As a mathematical model for the profile, a sum of several Pearson VII lines was used (Equation 1). The Pearson VII function provides the possibility of using a wide variety of line shapes – from Gaussian to Lorenzian, and beyond [16].

$$A(\nu) = \sum_{i=0}^{n} \frac{A_{i,0}}{\left[1 + 4Z_i^2 \left(2^{\frac{1}{m_i}} - 1\right)\right]^{m_i}} \qquad (1)$$

where:

ν = wavenumber
$A_{i,0}$ = absorbance in the center of peak i
Z_i = $(\nu - \nu_{i,0})/H_i$
$\nu_{i,0}$ = peak position of peak i
H_i = halfwidth of peak i
m_i = tailing factor of peak i
n = number of peaks

Trained to recognize $\nu_{i,0}$ in overlapping peak patterns, the ANN uses the first and second derivatives of $A(\nu)$ to predict the presence or absence of a peak at a particular ν. It is important to realize that the relation obtained after training was not derived mathematically, but was actually learned by the ANN.

The source code of the ANN program was written in Turbo Pascal and runs on an 80486 computer under MS DOS. Error backpropagation was used as a learning rule [18]. The network comprises four input nodes, one hidden layer with two nodes, and one output node; more hidden layers and/or more hidden units did not noticeably improve performance. The input nodes represent the following factors:

Node1: sign change of $dA(\nu)/d\nu$

0: no sign change
−1: sign change from − to +
1: sign change from + to −

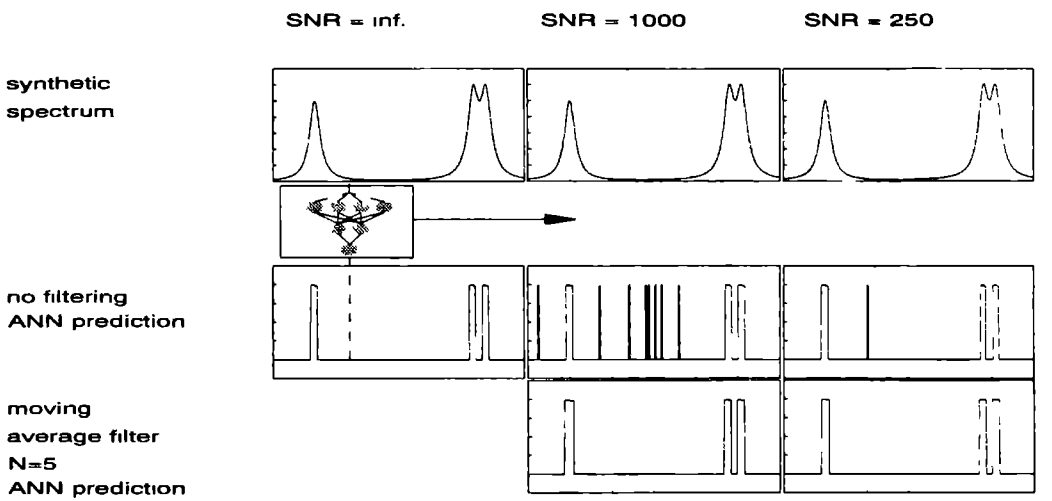

Figure 1: A trained ANN is used to detect peaks by scanning the profile. The ANN employs derivatives, and is therefore sensitive to noise. Noise is adequately reduced by applying a simple, moving average, filter to the original profile.

Node2: value of $d^2A(\nu)/d\nu^2$

-1: negative or zero

1: positive

Node3: sign change of $d^2A(\nu)/d\nu^2$

0: no sign change

-1: sign change from $-$ to $+$

1: sign change from $+$ to $-$

Node4: value of $dA(\nu)/d\nu$

-1: negative or zero

1: positive

Since the ANN employs derivatives, the method is sensitive to the SNR. The ANN was trained under ideal circumstances: no noise or baseline distortions were present.

The predictive capability of the trained ANN was assessed through several synthetic peak patterns with a different degree of SNR (Figure 1). It follows that the ANN does not only detect the peak maximum, but also predicts a region which corresponds to the halfwidth of the peak. Since the ANN only triggers on sign changes of first and second derivatives, it reacts irrespective of the signal amplitude. Although the training set was composed of synthetic spectra without noise, the ANN was able to detect peaks in noisy synthetic peak patterns and in experimental peak patterns, provided that some noise was removed; to that end, a simple filter sufficed.

To illustrate this, a prediction of the peak positions in an experimental X-ray diffractometer scan of poly(ethylene naphthalene-2,6-dicarboxylate), shown in Figure 2, was made. To increase the SNR, a moving average block filter was used with a window size of 5 data points. The diffractometer scans consisted of 470 data points. Note that the shoulder on the left-hand side of the rightmost bell was detected as a peak. The presence of this peak was confirmed by unit cell studies as a $\beta(200)$ reflection [2]. (Estimates for m_i of the Pearson VII line cannot be obtained by way of the ANN. For $A_{i,0}$, only an upper limit, $A(\nu)$, can be given.)

The performance of the ANN as a peak detection technique has not yet been studied intensively in combination with other methods. We are confident that such an approach is likely to improve the results and extend the scope of application.

2.2 Disregarding the baseline

When the original spectrum comprises many peaks with a large proportion of peak overlap, baseline detection is usually impossible. Only when data points are available in which no con-

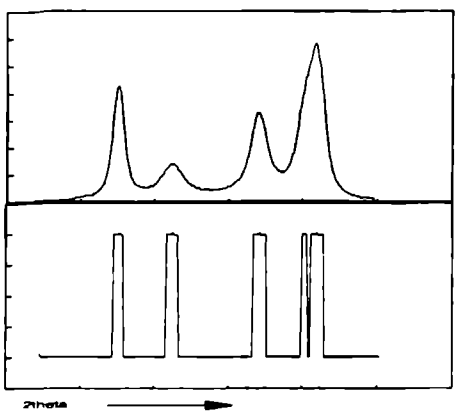

Figure 2: Scanned profile of an experimental X-ray equator diffractogram using the ANN.

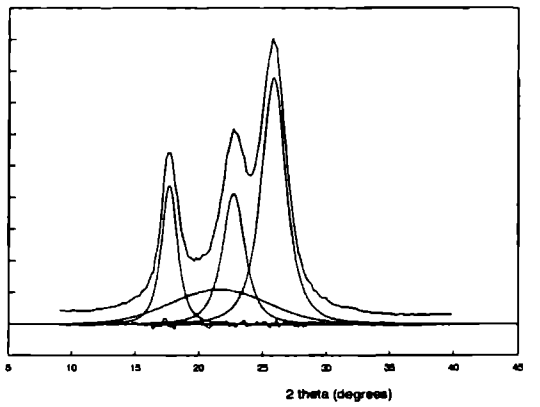

Figure 3: SD fit of an X-ray equator diffractogram of PET, using a four-line model. From the residual spectrum shown, it follows that the fit closely matches the profile.

tribution from peaks are present, the baseline position can be fairly well estimated by means of fitting, provided that these data points are situated at different positions on the ordinate of the spectrum. However, these ideal circumstances do not apply to complex spectra, e.g. infrared spectra. Added to the fact that the number of bands or peaks is not known, as a consequence of many overlapping bands, the baseline position can not be estimated. In this case, one might consider fitting the baseline together with the peak profile. In practice, however, this can give rise to ambiguous fitting results or to ending up in a local optimum, as baselines interact strongly with the tailing factors of the peaks.

In the cases we scrutinized so far, estimates of experimental spectra of X-ray diffractometer scans can be fairly easily obtained, because the peaks do not interfere with a baseline: no diffraction patterns were present at low angles ($< 9°$) and at high angles ($> 30°$) on the equator. An exponential baseline is assumed to compensate for the scattered light from the unrefracted X-ray beam. Research on more complicated spectra - *viz.* infrared spectra – to determine baseline positions is planned for the near future.

3 Curve fitting with steepest descent methods

In our laboratory, various curve fitting problems have been tackled successfully with a Gauss-Newton steepest descent method, or SD for short. Some specific cases involve: X-ray diffractometer scans of poly(ethylene terephthalate) (PET) yarns and nylon 6, as well as infrared spectra of PET.

In brief, the SD minimizes $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$ in non-linear least-square problems. The method is extended with a Marquarts technique to enforce convergence, and a technique for handling constraints [24, 3, 6, 7, 8]. Details are provided in the Appendix.

From peak positions, halfwidths, and peak intensities, derived from X-ray diffractometer scans, physically relevant parameters such as apparent crystallite sizes, lattice parameters, and amounts of crystalline material of a sample can be directly calculated. To obtain sufficiently accurate estimates for the various spectral parameters, a non-linear curve fitting routine is paramount. For relatively simple diffractometer scans, or diffractograms, the SD works well. It operates fast and is not very sensitive to initial estimates. As an illustration, an SD fit of an X-ray equator diffractometer scan of PET, using four Pearson VII lines, is shown in Figure 3. The residual error lies in the order of magnitude of the experimental noise.

In complex diffractograms, however, the initial estimates for the parameters in the model need to be accurate, otherwise the SD search is likely to end up in a local optimum, as pointed out above and shown shortly. Moreover, we noticed that as the complexity of the spectra increases, more

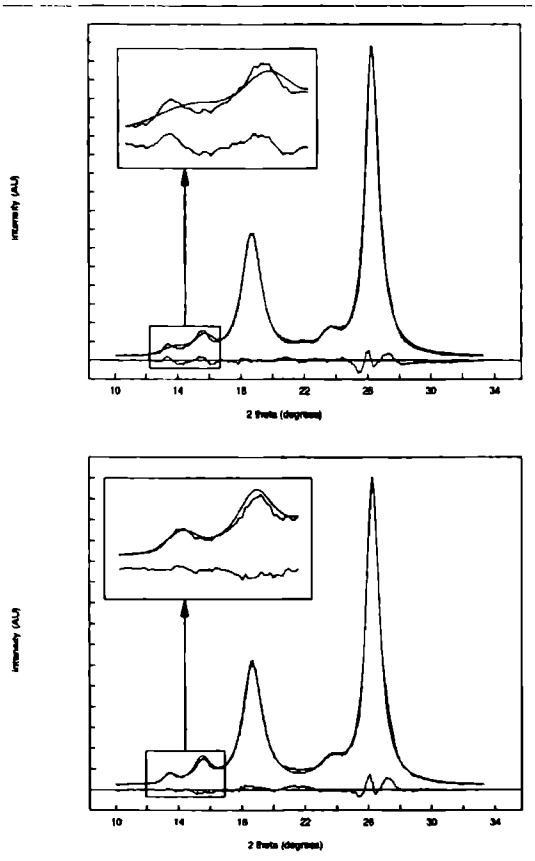

Figure 4: SD fits of two independently measured X-ray equator diffractograms of PEN, using a seven-line model. In both SD runs, the same initial estimates were used. Nonetheless, from the residual spectra, shown magnified, it follows that the search ends up in different optima; this indicates that SD search is sensitive to noise.

time is required for the development of a generally applicable model. For instance, in developing a six-line model of nylon 6 yarns, containing large γ crystals in the presence of α, it became apparent that inscrutable constraints on form factors and constant ratio of totally diffracted radiation have to be incorporated into the model in order to achieve acceptable results with the SD [6]. From recent studies on the morphology of poly(ethylene) naphthalate (PEN), it became apparent that even more accurate initial estimates are needed when the SD is used [21]; it was not possible to formulate constraints to decrease the dimensionality of the optimization problem.

Figure 4 depicts SD fits of two independently measured X-ray equator scans of PEN, using a seven-line model; six lines were detected by ANN peak tracing over a series of PEN yarns, and one extra line was added to compensate for amorphous scattering. The scans differ only in the constitution of the noise. Both fits are represented by dashed lines. The initial estimates for the peak parameters are listed in Table 1. The optimization criterion was a minimum residual variance. Although convergence was reached in both runs, for both samples a different fit (in terms of halfwidths, peak position, etc.) was obtained, regardless of the fact that the initial estimates were the same in both cases. This illustrates that small errors in the data caused by noise can be amplified to give large errors in the parameter values of the ultimate model. The fact that convergence was reached in both cases, confirms that SD search tends to end in local optima, or that there is a significant model error which caused ambiguous fitting.

Reflection	A_0	θ_0	H	$2-\frac{1}{m}$
Peak 1 $\beta_{(100)}$	0.04	13.47	0.85	1.98
Peak 2 $\alpha_{(010)}$	0.10	15.52	1.09	1.38
Peak 3 $\beta_{(020)}$	0.60	18.68	1.41	1.61
Peak 4 $\alpha_{(100)}$	0.08	23.97	1.99	1.69
Peak 5 $\beta_{(200)}$	0.88	26.20	0.85	1.82
Peak 6 $\alpha_{(-100)}$	0.20	26.50	1.41	1.17
Peak 7 α_{amorph}	0.10	20.50	11.0	1.98

Table 1: Values obtained by the ANN for the peak parameters in a seven-line Pearson VII model as an initial estimate for the SD.

4 Curve fitting with genetic algorithms

It is generally not desirable to force an optimization method to a solution by imposing constraints when the problem is actually not overdimensionalized, i.e. when a meaningful combination of values for all parameters exists that leads to a residual fit in the order of magnitude of the noise. From experience we know that the imposition of constraints is inevitable when steepest descent methods are used for high-dimensional curve fitting problems. This motivated us to develop a fitting procedure that is less sensitive to local optima. This procedure is based on genetic algorithms.

Genetic algorithms comprise a powerful and increasingly adhered search methodology which embraces principles of Darwinian evolution. They are especially suitable for complex, large-scale optimization; for more details about their concepts and operation, the reader is referred to [5, 13, 14].

Genetic algorithms are generally praised for their insensitivity to ending up in local optima, i.e. for their tendency to approach the globally optimal solution irrespective of diverse starting conditions. We therefore say that genetic algorithms are robust, i.e. feature a good search accuracy. In contrast, the search precision of genetic algorithms is poor: upon replicating runs, there is a considerable spread in the end solutions, despite the good search accuracy; that is, only the average end solution may generally be considered a reliable estimate of the global solution. Importantly, opposite properties often apply to traditional local optimization techniques, such as the aforementioned SD: a good search precision and a poor search accuracy, as pointed out above. Therefore, better overall performance can be attained in a sequential combination, where the genetic algorithm generates a "best guess" that serves as a starting point for subsequent improvement (refinement) by the local technique. Such and other hybrid techniques are important in that the constituent methods mutually supplement each other to achieve enhanced overall performance.

4.1 Hybridization for enhanced search

We consider three basic strategies to hybridization of genetic algorithms: pre-hybridization, post-hybridization and self-hybridization.

Pre-hybridization is concerned with finding an initial estimate for a genetic algorithm: in our case initial values for the Pearson VII model parameters. Even when this estimate is considerably inaccurate, which is most often the case, the genetic algorithm generally still performs well, as we will show shortly. In our application, the rough initial estimate is obtained from the aforementioned ANN for peak detection. Technically, an initial estimate for a genetic algorithm is called a working point. It forms the center of the real space in which the genetic algorithm searches. This space is also called the search volume, and its dimensions are specified by the user. The value ranges for the model parameters thus define the dimensions of the search volume in our application, i.e. their lowbounds and highbounds.

For each model parameter, only a limited number of values, or levels, specified by the user, are considered by a genetic algorithm. For convenience, these levels are chosen equidistantly. They amount to a search grid in the search volume. The mesh sizes that characterize the search grid dictate the search precision that can be attained by the genetic algorithm concerned. Randomly selected nodes on the search grid comprise the starting point of the search.

Post-hybridization is concerned with the improvement (refinement) of the end-solution found by a genetic algorithm. To that end, a local search method is used, e.g. the SD can be used. In this way, the local search constitutes a "remedy" for the poor search precision as a shortcoming of a genetic algorithm.

Self-hybridization is a strategy wherein a genetic algorithm is pre- or post-hybridized with another, differently configured genetic algorithm. We implemented this approach as a chain of genetic algorithms wherein each passes its result as the working point of the next in the chain. Thereby, in each step the search grid was kept invariant and the search volume was pruned; for further details on this strategy, the reader is referred to Chapter 2 (Section 8.3.4) in this thesis. In this way, one can accomplish significant reductions in the overall convergence time, such as to gain search precision and to preserve good search accuracy, simultaneously. The first few genetic algorithms in the chain are deliberately not run up until convergence, as this is not needed for the next.

4.2 Representation and search heuristics

Any search technique produces candidate solutions in an iterative way. These are guesses at the true solution of the problem concerned. A candidate solution is represented as a string (vector) of proposed values for the unknown parameters. Any string is evaluated by an objective function to assess its quality, i.e. its likeliness that it represents the true solution. A string may be modified by the search heuristics in an attempt to represent a better candidate solution. A distinguishing feature of genetic algorithms is that their evolution-

ary search heuristics manipulate a population of strings. In each iteration, or generation, the population is replaced by a new population of equal size. The new population is created by the evolutionary search heuristics in two steps. In the first step, strings in the current population are selected and copied at rates proportional to their quality, until the new population thus created is completed. The rationale for this is to obtain a new population that is expectedly better on average. Next, in order to reach potentially better candidate solutions, the strings in the new population are modified to some controlled extent. The generation cycle is closed through replacement of the current population by the new population.

An appropriate representation must be chosen, i.e. one which enables the search heuristics to impose modifications in such a way that the search efficiency is high. For our application (and numerous other applications), bitstrings, i.e. binary strings, the best representation for theoretical reasons [5]; we adopted such a representation. A bitstring may be regarded as a concatenation of bitfields: juxtaposed segmental bitstring parts that correspond with the respective unknown parameters of the problem. When an unknown parameter is encoded by a bitfield of B bits, then the value range of the unknown parameter is effectively subdivided into 2^B levels; thus, larger B values amount to a finer meshed search grid.

Many genetic operators exist for the modification of binary strings. We used **B_UX** (uniform binary recombination) and **B_M** (uniform binary mutation).

B_UX is iteratively applied, with probability P_r in each iteration, to twosomes of bitstrings obtained after randomly pairing the strings in the population. For each successful trial, it swaps with probability P_s each pair of bits that are at the same position. Thus, in brief, **B_UX** is parameterized by P_r (recombination probability) and P_s (swap probability). Important advantages of **B_UX** are: maximum exploratory power and positional unbiasedness; these properties are crucial to achieve good performance when the problem is highly non-linear, i.e. when many of the problem parameters are strongly correlated [14], as is the case in curve fitting.

B M inverts each bit in the population with probability P_m. Thus, in brief, **B_M** is parameterized by P_m (mutation probability).

4.3 Objective function

The evaluation of a bitstring proceeds as follows. First, the bitstring is decoded into a string of real values for the unknown parameters; a detailed description of this procedure can be found in [14]. The real values are then passed to the objective function. It calculates a spectrum according to the Pearson VII model and subsequently compares this spectrum with the known experimental spectrum to derive a measure of (dis)similarity. Two criteria that seem sensible are RMS (root mean square, a measure of dissimilarity) and CORR (correlation coefficient, a measure of similarity). Both criteria have shortcomings, though. For instance, CORR is not fully consistent with the purposes of curve fitting because it quantifies ratios rather than differences between spectra. RMS, on the other hand, is consistent but tends towards "plateau" optima on the RMS-landscape; this follows from "indecisive" search behavior observed near optima and may also be appreciated intuitively by considering small lateral perturbations in a spectrum. In order to obtain a criterion that is both consistent and leads to a structurally more pronounced landscape, we defined the bi-criterion $\text{ERROR} = \text{RMS}/(1 + \text{CORR})^2$ (a measure of dissimilarity). Empirically, we established that the minimization of ERROR leads to a significantly better performance than the minimization of either of its constituent criteria separately.

4.4 Configuration

The configuration is specified in a separate input file. Instead of providing a transcript of this file here, we merely summarize its key entries here:

Population size	: 100
Selective reproduction	: fitness-proportional
Fitness scaling mode	: sigmoid
Segregation degree	: 3.0
Fraction elitism	: 0.05
Recombination mode	: **B_UX**
P_r	: 0.85
P_s	: 0.3
Mutation mode	: **B M**
P_m	: 0.01
Encoding resolution	: 9 bits (512 levels)
Decoding mode	: Gray binary

For a more detailed explanation of the configurational terminology, the reader is referred to [14, 12], or Chapters 2 and 4 in this thesis.

4.5 Software and hardware

The routines that comprise a genetic algorithm fall apart into two parts: a domain-dependent part (concerning the problem representation and the objective function) and a domain-independent part (concerning the evolutionary search heuristics). By "domain-independent" routines are meant routines that can be used for other domains as well; for our application these were obtained from the software library GATES [15, 12]. In conformity with this library, the domain-dependent routines and data structures were programmed in C (ANSI standard) for speed and portability. The integration of all routines resulted in CFIT, or "GA" here: the executable application program for genetic curve fitting [11]. The GA was run under MS DOS on an 80486 computer.

5 Comparison of GA and SD

The two diffractograms that were fitted with the SD (Section 3), were also fitted with the GA, using the same seven-line mathematical model. The ANN was used to determine the search volume for the GA (pre-hybridization). This search volume is listed in Table 2.

The search volume obtained with the ANN turned out to be too small to achieve significant further improvement of performance through self-hybridization of the GA. However, we believe that self-hybridization will prove its usefulness when more complex spectra are concerned.

Post-hybridization of the GA by the SD method, too, did not improve performance significantly. Here, again, we believe that improvements will be achieved when more complex spectra are concerned.

Figure 5 shows the fitting results of the GA for both experimental diffractograms. In contrast with the results of the SD (Figure 4), the residual variance lies in the order of magnitude of the experimental noise, hence the possibility of a significant model error can be ruled out.

Furthermore, as shown in Figure 6, the GA finds compatible values for all halfwidths in both

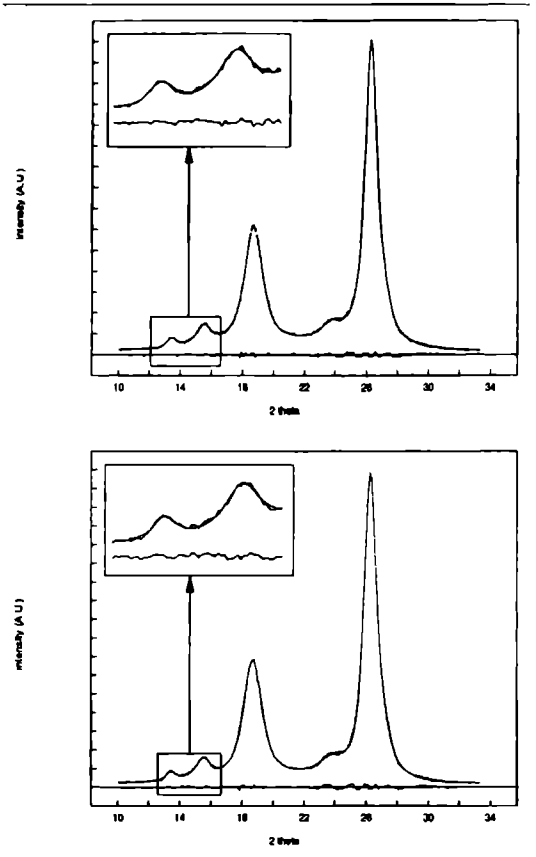

Figure 5: GA fits of two independently measured X-ray equator diffractograms of PEN, using a seven-line model. In both GA runs, the same initial estimates were used. From the residual spectra, shown magnified, it follows that both fits closely match the profile; this indicates that GA search is insensitive to noise.

diffractograms, whereas the SD produces significantly different values for the halfwidths of the poorly resolved peaks 1 and 6. (The standard deviations were calculated from the generalized inverse of the Jacobian matrix multiplied by the residual variance [1]; see the Appendix.) This is another indication that the GA is not sensitive to noise.

To show that this is also true for a more general case, the following experiment was set up. Ten PEN yarns were synthesized, each for a different value of process parameter ξ, keeping other process conditions fixed. X-ray equator diffractograms of the yarns were recorded. These scans

Reflection	Bound	A_0	θ_0	H	$2-\frac{1}{m_i}$
Peak 1 $\beta_{(100)}$	low	0.00	12.63	0.00	1.00
	high	0.13	14.33	2.71	1.98
Peak 2 $\alpha_{(010)}$	low	0.00	14.63	0.17	1.00
	high	0.28	16.33	1.87	1.98
Peak 3 $\beta_{(020)}$	low	0.21	17.82	0.54	1.00
	high	0.61	19.52	2.24	1.98
Peak 4 $\alpha_{(100)}$	low	0.00	23.29	1.39	1.00
	high	0.33	24.14	3.09	1.98
Peak 5 $\beta_{(200)}$	low	0.57	25.82	0.33	1.00
	high	1.17	26.88	1.52	1.98
Peak 6 $\alpha_{(-100)}$	low	0.00	26.45	1.10	1.00
	high	0.55	27.06	1.86	1.98
Peak 7 $\alpha_{\text{amorph.}}$	low	0.00	20.30	4.25	1.98
	high	0.15	23.70	12.8	1.98

Table 2: Search volume, i.e. lowbounds and highbounds, obtained by the ANN for the parameters in a seven-line Pearson VII model as an initial estimate for the GA.

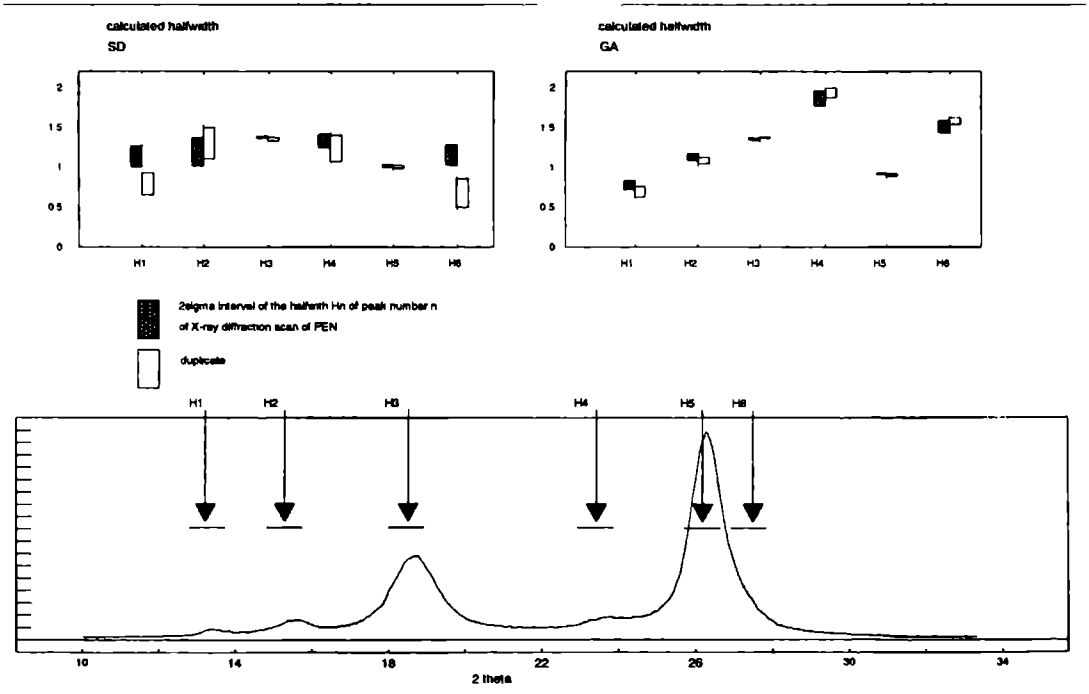

Figure 6: Comparison of the GA and the SD in estimating halfwidths (H_i, $i = 1, \cdots, 6$), in duplo. For peaks that are not well-separated (1 and 6), the SD produces quite different results, whereas the GA produces comparable results.

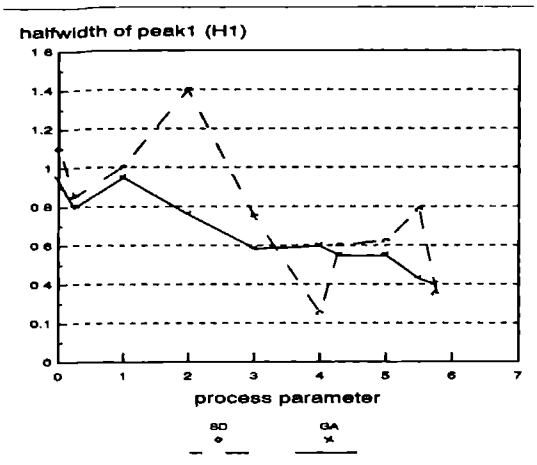

Figure 7: Comparison of the GA and the SD in estimating the halfwidth of the first reflection (H_1) in univariate relation with a process parameter.

were all fitted with the seven-line model, using the SD and the GA, respectively, as described above. In Figure 7, the halfwidth found for the first poorly resolved peak is plotted against the process parameter ξ, for SD and GA, respectively. Since this relation is univariate, one would expect that it is smooth. Instead, however, the SD produced chaotically scattered halfwidths. Thus, in spite of a fairly low error of fit, it was not possible to find a smooth empirical relation between the halfwidth of the first peak and ξ. This is another indication that the SD is sensitive to local optima. In contrast, the GA passed this test much better.

Table 3 lists some properties of the SD, the GA, and the so-called exhaustive search method, or ES, for the aforementioned seven-line diffractogram of PEN. In general, for any optimization technique applied to an ill-conditioned problem, there is no guarantee that the search convergences into a global optimum, except for the ES. However, the running time of the ES is exponentially proportional to the number of peak parameters; in practice, this means that reasonable time limits are exceeded in most cases. The SD and GA both require considerably less computation, but are at risk to miss the global optimum, i.e. to run into a local optimum. However, this risk is much smaller for the GA, as it searches less locally than the SD, yet the running time of the GA is not many order of magnitudes larger, as it samples the search space intelligently; therefore, as we showed in our study, the result is more accurate and less sensitive to the choice of initial estimates. It was empirically established by Lucasius [11] that the running time of the GA increases roughly as the third power of the number of peaks. In the case studied here, an acceptable solution is found within approximately 35 minutes (real-time) under MS DOS on an 80486 computer, and 8 minutes (real-time) under UNIX on a SUN Sparc 2 work station. The running time of the SD was only two minutes (real-time) per run under MS DOS on an 80486 computer, but it should be mentioned that at least 30 different runs were performed to achieve the result listed in Table 3, which was the best among all trials and evidently still of poor quality.

6 Final remarks and outlook

In fitting complex spectra, the steepest descent approach is likely to fail, unless very accurate initial estimates for the model parameters are given, which is often impossible in practical cases. Using X-ray equator diffractograms of poly(ethylene) naphthalate (PEN) yarns in a pilot study, we have shown that genetic algorithms are considerably less sensitive to initial estimates for, and noise in, the curve fitting problem, as they are generally less sensitive to local optima. Also, we have shown that the running time of this genetic algorithm lies within reasonable time constraints, especially when a rough initial estimate is provided by an artifical neural network trained for peak finding. This methodology, which thus comprises two promising forms of natural computation, will be further scrutinized in future research aimed at fitting spectra more complex than X-ray equator diffractograms: infrared spectra.

In brief, the following facts motivated us in conducting the study presented in this article. The steepest descent method was formerly applied successfully in our laboratory for fitting X-ray equator diffractogram of poly(ethylene terephthalate) (PET) yarns, but many constraints had to be imposed to enforce convergence towards acceptable solutions. These constraints are difficult to guess at, as they are often not physically grounded. This generally renders the formulation of contraints a time-consuming task, or even practically infeasible when complex spectra are concerned; for instance, in fitting infrared spectra of PET, typically about

Search strategy	Residual SS fit 1	Residual SS fit 2	Converged Solution	CPU time	Optimum type
ES	minimal	minimal	—	10^{50} min.	global
GA	0.92	0.89	yes(?)	35 min.	global(?)
SD	9.1	6.4	yes	2 min.	local

Table 3: Comparison of curve fitting by the GA, the SD, and the ES, in duplo.

100 band parameters are involved. Genetic algorithms are attractive in that they search large, complex problem spaces efficiently and productively without the need to impose strong domain assumptions.

Appendix

Consider the estimation of n unknown parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots \theta_n)$ in a non-linear mathematical model of the form:

$$y = f(x; \boldsymbol{\theta}) \tag{2}$$

where y is the dependent variable, or response, and x is the independent variable, or factor. To allow a solution strategy, the parameters of this model (Equation 2) must be bound on both sides by constants:

$$a_j \leq \theta_j \leq b_j, \qquad j = 1, 2, \ldots, n$$

The parameter estimates are obtained by means of a least squares fit to m data points (x_i, y_i), i.e. a set of parameters θ_j^* is determined such that the sum of squares $F(\boldsymbol{\theta}^*)$ is minimal:

$$F(\boldsymbol{\theta}^*) = \min_{\boldsymbol{\theta}} \sum_{i=1}^{m} (y_i - y(x_i; \boldsymbol{\theta}))^2$$

The minimization of this function is performed by the Gauss-Newton method. It is extended with a Marquarts technique to enforce convergence, and a technique for handling constraints [24, 3]. This process is iterative, starting with a feasible initial estimate between the boundary constants:

$$\boldsymbol{\theta}^0 = (\theta_1^0, \theta_2^0, \ldots, \theta_n^0)$$

The sequence of new estimates obtained through the iteration is such that the sum of squares gradually diminishes:

$$F(\boldsymbol{\theta}^{k+1}) < F(\boldsymbol{\theta}^k) \qquad k = 1, 2, \ldots$$

This process stops upon convergence, assessed by at least one of the following criteria:

- both the parameters and the sum of squares have been determined with a given accuracy;
- the sum of squares does not change significantly (in terms of the relative machine precision).

The variance/covariance matrix of the model parameter is calculated from the Jacobian matrix, $\mathbf{J}$, as follows:

$$(\mathbf{J}^T\mathbf{J})^{-1} s_r^2$$

where [1] $s_r = SS_{resid}/(m-n)$.

Acknowledgments

We acknowledge Akzo Research Laboratories Arnhem for financial support of this work. Furthermore, we are grateful to E.A. Sjerps-Koomen and C.J.M. van den Heuvel for providing the X-ray diffractograms of PEN.

References

[1] E.M.L. Beale. Confidence regions in non-linear estimation. *J. Roy. Stat. Soc.*, B-22:41–76, 1960.

[2] S. Buchner, D. Wiswe, and H.G. Zachmann. Kinetics of crystallization and melting behaviour of poly(ethylene naphthalene-2,6-dicarboxylate). *Polymer*, 30:480–488, 1989.

[3] N.R. Draper and H. Smith. *Applied Regression Analysis.* Wiley, New York, NY, 1967.

[4] S.R. Gallant, S.P. Fraleigh, and S.M. Cramer. Deconvolution of overlapping chromatographic peaks using a cerebellar model arithmetic computer neural network,. *Chemom. Intell. Lab. Syst.*, 18:41–57, 1993.

[5] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[6] H.M. Heuvel and R. Huisman. Five-line model for the description of radial X-ray diffractometer scans of nylon 6 yarns. *J. Polym. Sci.*, 19:121–134, 1981.

[7] H.M. Heuvel and R. Huisman. Infrared spectra of poly(ethylene terephthalate) yarns. Fitting of spectra, evaluations of parameters, and applications. *J. Appl. Polym. Sci.*, 30:3069–3093, 1985.

[8] C.J.M. van den Heuvel, H.M. Heuvel, W.A. Faassen, J. Veurink, and L.J. Lucas. Molecular changes of PET yarns during stretching measured with reo-optical infrared spectroscopy and other techniques. *J. Appl. Polym. Sci.* In press.

[9] R.S. Jackson and P.R. Griffiths. Comparison of Fourier self-deconvolution and maximum likelihood restoration for curve fitting. *Anal. Chem.*, 63:2557–2563, 1991.

[10] J.K. Kauppinen, D.J. Moffatt, M.R. Hollberg, and H.H. Mantsch. A new line-narrowing procedure based on Fourier self-deconvolution, maximum entropy, and linear prediction. *Applied Spectroscopy*, 45:411–416, 1991.

[11] C.B. Lucasius, A.P. de Weijer, L. Buydens, and G. Kateman. CFIT: A genetic algorithm for survival of the fitting. *Chemom. Intell. Lab. Syst.*, 19:337–341, 1993.

[12] C.B. Lucasius and G. Kateman. GATES towards evolutionary large-scale optimization: A software-oriented approach to genetic algorithms. Part 1. General perspective. Part 2. Toolbox description. *Computers & Chemistry*. In press.

[13] C.B. Lucasius and G. Kateman. Understanding and using genetic algorithms. Part 1: Concepts, properties and context. *Chemom. Intell. Lab. Syst.*, 19:1–33, 1993.

[14] C.B. Lucasius and G. Kateman. Understanding and using genetic algorithms. Part 2: Representation, configuration and hybridization. *Chemom. Intell. Lab. Syst.* In press.

[15] C.B. Lucasius and G. Kateman. *GATES: Genetic Algorithm Toolbox for Evolutionary Search, Software library in ANSI C.* Laboratory for Analytical Chemistry, Catholic University of Nijmegen, Nijmegen, January 1991.

[16] K. Pearson. *Phil. Trans. A.*, 186:343, 1895.

[17] J.A. Pierce, R.S. Jackson, K.W. van Every, P.R. Griffiths, and G. Hongjin. Combined deconvolution and curve fitting for quantitative analysis of unresolved spectral bands. *Anal. Chem.*, 62:477–484, 1990.

[18] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[19] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, Volume 1. MIT Press, Cambridge, MA, 1986.

[20] A. Savitzky and M.J.E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.*, 8:1627 - 1639, 1964.

[21] E.A. Sjerps-Koomen. Internal report. Technical Report CDSI92011, Akzo Research Laboratories Arnhem, Arnhem, 1990.

[22] J.R.M. Smits, L.W. Breedveld, M.W.J. Derksen, and G. Kateman. Pattern classification with artificial neural networks: Classification of algae, based upon flow cytometer data. *Anal. Chim. Acta.*, 258:11-25, 1992.

[23] B.G.M. Vandeginste and L. De Galan. Critical evaluation of curve fitting in infrared spectrometry. *Anal. Chem.*, 47:2124–2132, 1975.

[24] C. de Weerd. Technical Report Internal Report CRSI90005, Akzo Research Laboratories Arnhem, Arnhem, 1990.

[25] A.P. de Weijer, L. Buydens, G. Kateman, and H.M. Heuvel. Neural network used as a soft-modelling technique for quantitative description of the relation between physical structure and mechanical properties of poly(ethylene terephthalate) yarns. *Chemom. Intell. Lab. Syst.*, 16:77–86, 1992.

[26] J. Zupan and J. Gasteiger. Neural networks: A new method for solving chemical problems or just a passing phase? *Anal. Chim. Acta.*, 248:1–30, 1991. Review article.

CHAPTER 10

CFIT: A Genetic Algorithm for Survival of the Fitting

Contents

Chemometrics and Intelligent Laboratory Systems, 19 (1993) 337-341
Elsevier Science Publishers B.V., Amsterdam

Software Description

CFIT: a genetic algorithm for survival of the fitting

C.B. Lucasius, A.P. de Weijer, L.M.C. Buydens and G. Kateman

Laboratory for Analytical Chemistry, Faculty of Science, Katholieke Universiteit Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands)

(Received 12 November 1992; accepted 9 December 1992)

Abstract

Lucasius, C.B., De Weijer, A.P., Buydens, L.M.C and Kateman, G., 1993. CFIT: a genetic algorithm for survival of the fitting. *Chemometrics and Intelligent Laboratory Systems*, 19: 337-341.

CFIT is discussed: an application program of a genetic algorithm dedicated to spectral curve fitting. CFIT's practical utility resides in several attractive properties of genetic algorithms that other techniques traditionally used for curve fitting apparently lack, inter alia robustness. Thus far, the results are promising. Since recently, CFIT is used at a chemical company.

INTRODUCTION

(The work presented here is an extension of [1].)

Peak deconvolution by curve fitting is an important technique among data reduction techniques used for the interpretation of experimental spectra. The procedure is based on a mathematical model assumed for the peaks in the spectra; we adopted a Pearson VII peak model, which encompasses, inter alia, pure Gauss forms, pure Lorentz forms and mixed forms thereof. Each peak is described by four model parameters — A_0 (height), X_0 (position), H (width at half height) and Z (tailing factor); the latter determines the Gauss/Lorentz proportion. The mathematical model is non-linear and since it cannot be solved analytically, a search procedure is required: iteratively, a set of values for the model parameters is proposed and in each step the model is used to calculate a spectrum that is compared with the known experimental spectrum; the comparison is done on the basis of some predefined dissimilarity criterion, or error function.

Before starting a curve fitting procedure, the user must specify the maximum number of peaks expected in the experimental spectrum. Evidently, this is a critical decision; in general, one stays at the safe side if the maximum is chosen too large (rather than too small), because this leaves open the possibility that zero heights are assigned to the excess peaks; on the other hand, the search is then complicated by ambiguities.

Correspondence to. Dr. L.M.C. Buydens, Laboratory for Analytical Chemistry, Faculty of Science, Katholieke Universiteit Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, Netherlands.

SEARCH METHOD [2,3]

All search techniques have in common that they basically aim to improve an initial estimate of the true solution. Our program CFIT (Curve FITter) uses a genetic algorithm to search the global minimum on the error landscape in the space spanned by the model parameters; the global minimum represents the true solution. Best performance is obtained when a proposed set of values for the model parameters is encoded as a binary string, or a bitstring. A population of bitstrings is maintained and updated iteratively. (The initial population is chosen randomly.) In each step, all bitstrings are evaluated by the error function and are assigned a reverse error value as a measure of performance, or fitness (in genetic jargon).

The updating procedure is inspired by principles of natural evolution according to Darwin. First, the strings are reproduced at rates proportional to their fitness, until the new population thus formed reaches the size of the current population; the idea behind this is Darwin's 'survival of the fittest': the new population is, on average, expected to perform better due to the selection 'pressure' exerted; in CFIT this comes down to survival of the bitstrings that give rise to the best fitting spectra calculated. Next, in order to reach potentially better estimates, the bitstrings in the new population are modified to some controlled extent by operators reminiscent of crossover and mutation as applied to biological chromosomes; these and other genetic operators are based on random events. The evolution cycle is closed by replacing the current population with the new population, and the next cycle is started.

EVALUATION

Each bitstring evaluation proceeds in basically the following steps.

First, the bitstring under consideration is decoded into a string of real values for the unknown model parameters; this is a computationally cheap step. Then, applying the peak model to these values, datapoints are calculated that constitute a proposed spectrum: these data points are for the same wavelengths as the data points that constitute the experimental spectrum.

Spectrum calculations amount by far to the computationally most intensive part of CFIT; the burden (time), T_1, for one such calculation is roughly proportional to (1) N_{peak}, the number of peaks assumed (because peaks are calculated separately before they are summed into a spectrum), and (2) N_{data}, the number of data points that comprise the spectrum:

$$T_1 \propto N_{peak} N_{data}$$

Next, the error function compares the calculated spectrum with the experimental spectrum on the basis of a bi-criterion, comprising RMS (the root mean square — a measure of dissimilarity) and CORR (the correlation coefficient — a measure of similarity): simultaneously, RMS is minimized and CORR is maximized. Thereby, CORR is basically used to pronounce the RMS landscape; indeed, the bi-criterion usually leads to better search performance than when RMS alone is used.

PROPERTIES

The practical utility of CFIT resides in some generally attractive features of genetic algorithms: robustness (relative to local search) and efficiency (relative to global search).

Local search techniques traditionally applied to tackle curve fitting problems have a notorious record of unreliability, related to the following coincidence: (1) many local minima besides the global minimum may be present on the error landscape, especially when strong peak overlaps occur in the spectrum to be fit; (2) as a rule, no accurate initial estimate of the solution can be given. Consequently, the search ends up in the nearest optimum, which is frequently local; under these circumstances a short convergence time becomes worthless.

A more robust search strategy demands the use of search techniques that employ non-local search heuristics, e.g., statistical 'cooling' algorithms (simulated annealing) or genetic algorithms. In general, non-local search requires more

computation than local search in order to converge, but the result is more accurate and less sensitive to the choice of the initial estimate; besides, nowadays computation is fast and cheap.

Relative to global search — that is, systematic scan of the legally proposable solutions in the search space — genetic algorithms are extremely efficient. This is a consequence of exponential search behavior — a property that follows from the so-called schema theorem (the mathematical framework of genetic algorithms); the next section illustrates this point.

PERFORMANCE [3]

The efficiency of CFIT can be appreciated by the following comparison with global search.

Suppose the task is to fit five peaks ($N_{peak} = 5$); thus, the search space is spanned by twenty model parameters (four per peak). Next, the range of values for the model parameters is digitized into 512 levels; this reflects a reasonable desired precision. The levels define a search grid in the twenty-dimensional search space. The number of nodes on this grid is the size of the search space: $512^{20} \approx 10^{54}$. This size is prohibitively large for global search using today's or tomorrow's fastest computers. CFIT, on the other hand, requires only 10^4 evaluations to converge towards an acceptable solution; this corresponds to approximately 10 min real-time on an IBM PC 80486 platform (33 MHz) for a spectrum with $N_{data} = 200$.

CFIT's running time, T_{cfit}, equals $T_1 N_{popl} N_{iter}$, where N_{popl} is the population size (the number of bitstrings in the population) chosen and N_{iter} is the number of iterations (the number of population updates) required to attain convergence. Hence,

$$T_{cfit} \propto N_{peak} N_{data} N_{popl} N_{iter}$$

Empirically, we obtained best performance figures for $N_{popl} \approx 20 N_{peak}$; thereby we noted that, roughly, $N_{iter} \propto N_{peak}$. Thus, T_{cfit} increases roughly as the third power of N_{peak}. This compares very favorably with the exponential rate at which the size of the search space ($512^{4N_{peak}}$) grows with N_{peak}.

Nonetheless, when T_{cfit} would exceed practical time constraints, special provisions need to be made. Attractive strategies to that end are for example those wherein some method improves the initial estimate for CFIT first — thereby basically reducing N_{iter}, thus reducing T_{cfit}; a method used for such prior improvement is called a pre-hybridizer, and the overall sequential hybrid searching system is called a hyphenated search method. Its overall running time is $T_{hybr} = T_{pre} + T_{cfit}$, where T_{pre} is the time that the pre-hybridizer is run. The aim of pre-hybridization is, of course, to obtain a reduction in T_{cfit} that is larger than T_{pre}; the optimal T_{pre} must be determined empirically.

We have obtained good results with the following pre-hybridizers.

(1) An artificial neural network trained for finding peaks.

(2) CFIT itself, using a version of the experimental spectrum in which every second data point is weeded out; this halves the convergence time at the expense of some accuracy in the new estimate; a more severe weeding strategy may be considered in order to achieve further reduction at more expense of accuracy (incidentally, genetic algorithms are generally touted for their robustness against missing data and noise).

(3) CFIT itself, in one run applied to the left and another run applied to the right half of the spectrum, assuming half of the total amount of peaks in each; this reduces the total convergence time by roughly a factor $\frac{1}{2^3} + \frac{1}{2^3}$, at the expense of some accuracy in the new estimate; a higher degree of windowing (segmental partitioning) may be considered in order to achieve further reduction at more expense of accuracy.

(4) CFIT itself, halted before convergence is attained; the idea behind this is to use the estimate as the center of a pruned (smaller) search volume in a next run of CFIT (see example below).

CFIT comes with a post-hybridizer for further hyphenation (sequential hybridization): a steepest descent algorithm that serves to improve (refine) CFIT's ultimate estimate. Now, the overall running time of the hybrid searching system is $T_{hybr} = T_{pre} + T_{cfit} + T_{post}$, where T_{post} is the time that

the post-hybridizer is run. The post-hybridizer can play a similar role as the pre-hybridizer in minimizing T_{hybr}.

USAGE

CFIT is started with command-line arguments, e.g. (with explanation).

```
CFIT domain.1 domain.2  2 2 2 2 0
  1437 b
```

(see Table 1).

When the above session is configured for, say, 30 iterations, then a next session of CFIT might be specified as for example:

```
CFIT domain.2 domain.3  2 2 2 2 30
  9514 b
```

(Note that `domain.2` is now the first argument.) In this way, the estimate provided by the first CFIT run is further refined by CFIT itself (see the preceding section), now starting at logical iteration 30.

On start-up, CFIT implicitly assumes the existence of a file called `control.ga`, which specifies values for the genetic control parametes; e.g. the crossover rate, the mutation rate, the selection rate, etc.

SOFTWARE AND HARDWARE

The routines comprising CFIT are dichotomized into two parts: a domain-dependent part (concerning the problem representation and the error function) and a domain-independent part (concerning the genetic search heuristics). By 'domain-independent' routines are meant routines that can be used for other domains as well; for our application these were obtained from the software library GATES [4,5]. The integration of all routines resulted in CFIT.

CFIT is presently available for MS DOS on IBM PCs and for UNIX on SUN Sparc stations; a provisional manual is available. The source code of CFIT is written in C for efficiency and can be ported to any platform that supports an ANSI-standard C compiler.

Since recently, CFIT is used routinely at ARLA (Akzo Research Laboratories Arnhem, Netherlands). Depending on the response on the present paper, it is our intention to commercialize this software product; please contact the authors for more details.

REFERENCES

1 A P de Weijer, C B Lucasius, L. Buydens, G Kateman, H M Heuvel and H Mannee, A new approach to curve-fit-

TABLE 1

Example of CFIT command line arguments and their explanation

`domain.1`	The name of an input file that contains information about the domain, inter alia the name of the input file that contains the spectrum to be fit, an initial estimate of the solution, the dimensions of the search volume around this estimate, the name of an output file to which the fitted spectrum will be written
`domain.2`	The name of an output file with the same format as `domain.1` and that specifies, inter alia the name of the input file that contains the spectrum that was fit, a new (hopefully better) estimate of the solution, the proposed pruned dimensions of the search volume around this estimate if `domain.2` would be used as the first argument in a next call of CFIT (see example)
`2 2 2 2`	Pruning factors for the search volume dimensions concerning the model parameter types A_0, X_0, H and Z, respectively
`0`	Logical iteration offset The physical iteration offset is always 0, but the logical iteration offset should be set at the last iteration where a previous run ended, to ensure that entries written to output files that monitor performance are appended consecutively
`1437`	Seed for the random generator used
`b`	Batch mode (optional)· interaction with the user is suppressed This is especially useful when many sessions (runs) are planned in sequence, called from a batch file

ting using natural computation, *Analytical Chemistry*, submitted for publication.
2 C.B. Lucasius and G. Kateman, Understanding and using genetic algorithms. Part 1. Concepts, properties and context, *Chemometrics and Intelligent Laboratory Systems*, 19 (1993) 1–33.
3 C.B. Lucasius and G. Kateman, Understanding and using genetic algorithms. Part 2. Representation, configuration and hybridization, *Chemometrics and Intelligent Laboratory Systems*, submitted for publication.
4 C.B. Lucasius and G. Kateman, GATES towards evolutionary large-scale optimization: a software-oriented approach to genetic algorithms. Part 1. General perspective, *Computers & Chemistry*, submitted for publication.
5 C.B. Lucasius and G. Kateman, GATES towards evolutionary large-scale optimization: a software-oriented approach to genetic algorithms. Part 2. Toolbox description, *Computers & Chemistry*, submitted for publication.

This program has been implemented by us and performs as described. H.M. Heuvel, Akzo Research Laboratories Arnhem, P.O. Box 9300, 6800 SB Arnhem, Netherlands.

CHAPTER 11

Sequential Assignment of 2D-NMR Spectra of Proteins Using Genetic Algorithms

Contents

Sequential Assignment of 2D-NMR Spectra of Proteins Using Genetic Algorithms

R. Wehrens,* C. Lucasius, L. Buydens, and G. Kateman

Laboratory for Analytical Chemistry, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Received November 23, 1992

The application of genetic algorithms to the problem of the sequential assignment of two-dimensional protein NMR spectra is discussed. The problem is heavily underconstrained since in most cases more patterns are available than amino acid positions, and uncertainties may exist in the preliminary assignments. The results indicate that relatively large amounts of errors may be present in the input data for the genetic algorithm while useful results may still be obtained.

INTRODUCTION

The bottleneck in the calculation of the three-dimensional structure of proteins from NMR data lies in the spectrum interpretation. Several months of an expert's time may be required to obtain a preliminary interpretation that has to be refined further. Therefore, attempts have been made to automate the process, at least to the extent that the expert is aided in the interpretation process.[1-6] However, no systems are known that are able to obtain a complete interpretation for proteins of reasonable size without human assistance. This is largely due to the anomalities that occur in every spectrum, such as missing peaks because of noise or insufficient resolution, and atypical patterns. The flexibility and robustness that is necessary to solve a problem of this complexity can only be found in an iterative process, where an expert is able to switch back and forth to pursue his ideas freely. Such an iterative process is difficult to implement, however, and therefore most systems aim at providing a best first guess. These results may then be validated manually and included in the next run of the system.

Most systems apply the so-called sequential assignment strategy,[7] in which the interpretation consists of three stages. In the first stage the spin systems of the separate amino acids (hereafter denoted as patterns) are identified in the COSY spectra. Uncertainties can arise whether a peak does or does not belong to a specific pattern. Some programs solve this uncertainty by retaining two versions of the pattern: one with and one without the doubtful peak. Another reason for the number of patterns to be larger than the number of amino acids is the presence of *glycine* amino acids in the sequence. Because of the two C_α protons in a glycine, two patterns instead of one pattern may be identified.

In the second stage, these patterns are assigned to amino acid types. This is done on the basis of characteristic features of the patterns. However, in most cases it will not be totally clear to what kind of amino acid a pattern should be assigned, and therefore some errors are inevitable in this stage, too.

Finally, the patterns are assigned to amino acids at specific positions in the sequence. This is done by identifying a list of possible pattern pairs for each pair of vicinal amino acids in the chain. For a combination of patterns to be valid for an amino acid pair, the types of the patterns must match the types of amino acids, and a sequential connectivity in the NOESY spectra should be found. The pairs of patterns found in this way are then combined to map to the complete amino acid sequence.

The last part is especially difficult, since an exhaustive search will often lead to combinatorial explosion. In this paper, we propose a new approach to the last step that we will call, for convenience, the sequential assignment step, using genetic algorithms.[8] Genetic algorithms (GA's) are problem solvers that are very powerful in searching large solution spaces, mimicking the principle of "the survival of the fittest". Candidate solutions are ranked according to some evaluation function that is written for the specific problem, and successful solutions are allowed to reproduce with other successful solutions. This way, after several generations, the population consists of solutions that have inherited the strong parts of their ancestors and that hopefully form a better solution to the problem. In the past, genetic algorithms have been applied to a wide variety of problems,[9-11] especially to those for which an analytical solution was not possible. The genetic algorithm that is applied here is part of a hybrid expert system, HIPS.[12] It must be stressed, however, that any other system may be used to provide the necessary input files. Because of the nature of the input needed for the genetic algorithm, it is to be expected that most other interpretation programs will be able to produce the necessary information. The GA will be discussed thoroughly in the following section.

The data sets that have been used to test the performance of the genetic algorithm have been obtained by the HIPS program, using COSY, NOESY, DQSY, and RCT spectra. They contain the following information:

(a) The sequence of amino acids.

(b) A list of combinations of vicinal amino acids from the sequence.

(c) For each combination of amino acids, a list of combinations of spin patterns that may match the amino acids.

A sample input file is shown partially in Figure 1.

Ideally, for each combination of amino acids, the number of possible pattern combinations should match the number of occurrences of the amino acid combination in the sequence. In such a case, sequential assignment would be trivial and could easily be performed manually. In practice, however, the problems posed are much larger. In expert problem solving, it is often necessary to reject one's own hypotheses and start anew. This is due to the inherent uncertainty in composing the patterns (does this peak belong to the pattern or not?) as well as in assigning the patterns to amino acids (can this pattern be caused by a valine?). As already stated, most spectrum

0095-2338/93/1633-0245$04.00/0

```
SEQUENCE INFO >
  size : 58
  aa's : R P D F C L E P P Y T G P C K A R I I R Y F Y N A K A G L
         C Q T F V Y G G C R A K R N N F K S A E D C M R T C G G A

PATTERN INFO >
  nr of patterns : 64
  patterns:
0   CYS-51
1   PATTERN-3
2   GLY-56
.
.
63  PROLINE-4

COMBINATION INFO >
  nr of aa combinations: 54

  G  A :
  nr of pattern combinations: 25
27  3
56  3
.
.
55  20

  C  G :
  nr of pattern combinations: 16
32  1
35  1
.
.
41  59

.
.
```

Figure 1. Sample input file (partially shown) of the BPTI data set, obtained with global settings (see text). Pattern identifiers like CYS-51 are optional and are used only for validation purposes here.

interpretation programs handle this by allowing for multiple assignments, but this will increase the number of possibilities in the sequential-assignment step significantly. Moreover, the number of potential sequential connectivities in the NOESY spectra may be quite large, and overlapping patterns may pose difficult problems, too. Thus, the number of pattern combinations may be expected to exceed the necessary number by a large amount. These types of errors may be called *false positive* (FP) errors. Furthermore, some pattern combinations that actually are present in the "correct" sequence may be missing: *false negative* (FN) errors. A sequential connectivity may fall below the noise level, or a pattern may be assigned to the wrong type of amino acid. In such a case, the "correct" pattern combination will not be present in the input file for the sequential assignment module.

Systems that perform an exhaustive search for the best sequence of patterns are not very robust against both types of errors. A large number of FP errors will cause the search to diverge to such an extent that it is completely intractable. FN errors may cause the search to pursue false directions in an early stage, yielding spurious results. The genetic algorithm may be expected to be more robust against both types of error. Superfluous information in the form of FP errors will cause a lot of suboptima to be present in the solution space, but the GA has a reputation of being capable of finding global optima, even in very difficult search spaces.[8] Missing pattern combinations (FN errors) will of course prevent the system from finding a complete solution, but since a complete candidate solution is evaluated at once, nothing prevents the GA from placing the correct patterns at the following positions.

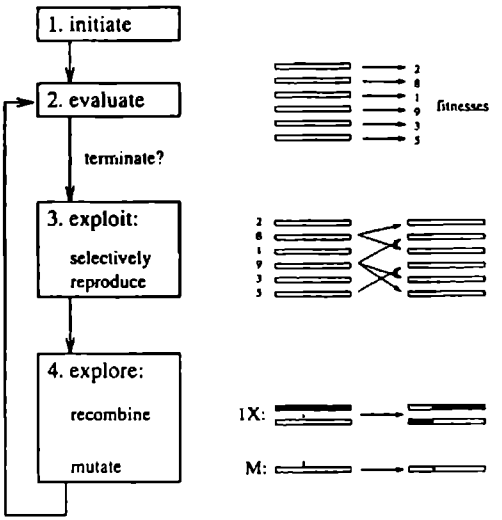

Figure 2. Flow chart of a genetic algorithm.

GENETIC ALGORITHMS

In this section, the fundamentals of genetic algorithms are reviewed briefly and the most important ideas and concepts are discussed. The principal idea is that of the survival of the fittest; a population of candidate solutions is evaluated, and only the best are allowed to reproduce. Each solution is represented as a bit string; each string evaluation yields a fitness. Solutions are ranked to their fitnesses and allowed to reproduce themselves with a probability proportional to their fitness. Reproduction takes place by means of the so-called crossing-over operator. In a crossover, random parts of the parent strings are combined to form a new child string. In this way, large parts of successful solution strings are combined to form new and hopefully even better solution strings. After reproduction, the parent strings are deleted to keep the number of strings constant. At the reproduction stage, some random mutations are introduced to keep the population from premature convergence. These concepts are illustrated in Figure 2. As the number of generations increases, and bad solutions are deleted, the fitness of the best member of the population may be taken as an indication of the success of the search. In Figure 3 a typical GA run is depicted. The evaluation criterion yields discrete fitness values that rapidly increase in the beginning of the search. After a while, an optimum is reached and the search can be terminated. The termination criterion is usually a maximum number of generations or a maximum fitness value.

Whereas the concepts are simple, a lot of small variations and extensions are possible. For instance, so-called two-points crossover operators may be defined that exchange parts of bit strings that are not terminal, and penalty functions may be included in the string evaluation in the case that the solutions become too much alike. Furthermore, crossover and mutation rates may be varied, along with population sizes, and scaling functions may be applied to the fitness. This makes a genetic algorithm a very flexible tool.

However, the most important issue in applying genetic algorithms is the representation of the problem. A suitable way must be found in which solutions can be represented as bit strings and in which they can be manipulated easily.

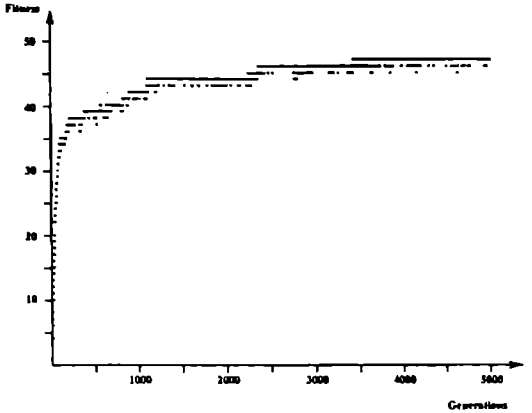

Figure 3. Performance plot of a genetic algorithm.

Furthermore, an appropriate evaluation criterion must be defined. If any of these elements is not optimal, the genetic algorithm will perform poorly. For numerical optimization problems, several established methods have been set up, and application of genetic algorithms in such cases is relatively straightforward.[9] However, subset selection and sequencing problems often require another set of primitives.[13]

Application to the Sequential Assignment. The sequential assignment problem is essentially a subset selection problem where also the sequence of the solution is of importance. In principle, N patterns should be mapped to M positions in the sequence (where $N > M$). The size of the search space is $N!/(N-M)!$. This number can be very large; as an example, one of our data sets consisted of a protein of 74 amino acids and 97 possible patterns, yielding a search space of approximately 10^{130} solutions! The genetic algorithm that is applied in this cases uses a specially developed subset encoding along with special crossover and mutation operators.[13] In short, each solution is represented as a permutation of the N possible elements. Only the first M elements are evaluated in the fitness function so that the subset selection takes place automatically. The crossover operator preserves position as much as possible: an element on a certain position in the parent string will be copied to the same position in the child string, whenever possible.[13] Mutation consists of random swapping of two elements on the bit string and is divided into a *reorder mutation* and a *trade mutation*. The former swaps two elements in the first M elements of the bit string; the latter swaps an element from the first M elements with an element of the last $N - M$ elements of the bit string.

The fitness criterion basically counts the number of pattern combinations in a candidate solution that are present in the list of possible pattern combinations (for each combination of vicinal amino acids). As an example, consider the amino acid combination GA (*glycine alanine*) at the end of the sequence in Figure 1. Twenty-five combinations of patterns are possible here, according to the input; whenever in a candidate solution one of these pattern combinations is at the last positions in the sequence, the candidate solution receives 1 point. In this way, a maximum fitness of $M - 1$ is possible. In that case, all pattern combinations match the combinations of amino acids to which they are mapped.

Throughout all experiments, the crossover rate was kept at a value of 0.8, the reorder mutation rate at a value of 0.02, and the trade mutation rate at a value of 0.01. The mutation rates are kept low to prevent the search from becoming random.

Table I. Characteristics of the Original Data Sets[a]

protein	settings	no. of patterns	no. of pattern pairs	no. of FN
E-L30	global	61	922	29
E-L30	optimal	61	1032	31
BPTI	global	64	1567	21
BPTI	optimal	63	1032	16
Tendami	global	97	4042	19
Tendami	optimal	90	2244	20

[a] E-L30 and BPTI both consist of 58 amino acids; Tendami consists of 74 amino acids. In the E-L30 data set, more than half of the "correct" pattern combinations are not present in the input file. The Tendami sets are difficult because of the large number of pattern combinations that are possible and the large number of superfluous patterns that have been found.

The values used for these parameters are in agreement with the optimal values found using an experimental design. The population size was usually set to 500, and a maximum number of generations of 3000 was used as a stop criterion. Fitnesses were scaled using a sigmoid function, to enable the genetic algorithm to make progress even in flat solution spaces.[13]

The programs were written using the GATES software[14] and were run on a SUN SPARC1 workstation. One run of 3000 generations usually took 4 h of real time.

RESULTS AND DISCUSSION

In this section, the results of a number of test cases with real and simulated data sets will be discussed. With these sets, we can determine the amount of incorrect superfluous, or missing data that the GA can handle in reaching a sensible solution. To be clear, it is not our claim that the GA will find the complete and correct sequence of patterns but rather a large part of the solution that then can be used to obtain the missing parts, either by hand or by a next iteration with the help of an expert system. The test cases serve as a means of illustrating this.

Data Sets. The data sets are derived from three proteins: E-L30, BPTI (bovine pancreatic trypsine inhibitor), and Tendamistat. Fabricated spectra of these proteins[5] were used in the assignment using the expert system HIPS.[12] In HIPS, the conclusions of the expert system can be evaluated using a database of solved cases, and suggestions can be done by the system to improve its performance. This way, several parameters in the expert system can be tuned to obtain optimal results. For each protein, two data sets were produced, one set in which the performance of HIPS was optimized for that particular protein, and one set in which global settings were used that were optimal for the total of the three proteins. These original sets are denoted "optimal" and "global", respectively. Thus, we obtained six realistic data sets (two for each protein), in which different amounts of FP and FN errors are present. These six data sets are described further in Table I. It should be noted that these sets have been obtained using the fully automatic mode of HIPS, and that the only manual intervention has been the training of HIPS to the three data sets to obtain a set of optimal settings for each protein and a set of global settings which yielded an overall maximal performance.

From these data sets, several others were constructed manually to test the abilities of the GA. First of all, the optimal data sets from Table I were pruned manually to reduce the number of possible pattern pairs. This is not an unrealistic situation since in real life the spectrum interpretation program will most probably be used by an expert who is capable of limiting the number of possibilities in an early stage. In this

Table II Characteristics of the Constructed Data Sets[a]

protein	characteristics	no of patterns	no of pattern pairs	no of FN
E-L30	optimal settings, pruned	61	218	30
	optimal settings, full	61	1063	0
	optimal settings, pruned, full	61	249	0
	global settings full	61	929	0
BPTI	optimal settings, pruned	63	352	16
	optimal settings full	63	1050	0
	optimal settings pruned, full	63	370	0
	global settings, full	64	1590	0
Tendami	optimal settings, pruned	90	520	21
	optimal settings, full	90	2244	0
	optimal settings, pruned, full	90	533	0
	global settings, full	97	4042	0

[a] In case the number of FN errors increases by pruning, a pattern that has been inserted for a missing assignment in literature has been pruned away

case, the pruning was continued for each combination of amino acids until less than ten possible pattern combinations were left As the "correct" solutions of patterns of all data sets used here are known, this knowledge was used as the pruning criterion, if a pattern in a combination did not match the amino acid type in that combination, the pattern combination was removed This pruning procedure ensures that the FP errors continue to be distributed evenly over all combinations of amino acids In case an amino acid has not been assigned in the literature, a pattern that is not part of the sequence will be inserted at random For the present purposes, they can be treated as if they were correct Furthermore, all original data sets, as well as the pruned data sets, were provided with the correct pattern pairs that were missing (column no of FN in Table I), again using the knowledge of the correct pattern sequence This yielded nine more data sets in which the fitness of the true solution is the highest fitness possible The data sets constructed in the above ways have been gathered in Table II

EXPERIMENTAL SECTION

In each experiment, five runs with the GA were done From each run, the solution with the highest fitness was selected If two or more solutions had the same fitness, the last one generated was selected An assignment of a pattern to a position is considered definite if in at least three of the five selected solutions the pattern is placed at that position An assignment is considered possible if this is the case with two of the five runs No attempt is made to check whether a pattern has been assigned to two places at once

Furthermore, each data set was used in four experiments that used a slightly different fitness evaluation function In the normal case, as has already been described, the number of correct pattern pairs present in the trial solution is counted In the three other evaluation functions, extra points are given if two correct pattern pairs follow each other That is, if the middle pattern of the three is possible in both pattern pairs The amount that is added in such a case is varied in this way we can see the influence of the evaluation on the eventual performance of the GA In the first experiment, 0 5 is added for each pair of correct pattern pairs, in the second 1, and in the third 2

The sets where the real solution has the highest fitness can be used to assess the sensitivity of the genetic algorithm to superfluous and misleading information, by validating its ability to reach a global optimum in a complex solution space

Table III. Mean Fitnesses and Standard Deviations of Fitnesses in Five Experiments for the Data Sets[a]

protein	characteristics	"correct" F	mean F	s_F
E-L30	global settings	27	34 2	2 4
	optimal settings	29	33 8	2 8
	pruned	28	27 8	1 9
	global settings, full	57	41 0	5 6
	optimal settings, full	57	39 8	4 5
	pruned, full	57	54 5	2 5
BPTI	global settings	37	42 8	1 6
	optimal settings	42	40 4	2 4
	pruned	42	42 8	2 3
	global settings, full	57	45 8	2 7
	optimal settings, full	57	42 0	2 5
	pruned, full	57	49 2	7 9
Tendami	global settings	55	62 4	2 3
	optimal settings	54	61 0	2 9
	pruned	53	47 8	4 5
	global settings, full	73	66 2	3 0
	optimal settings, full	73	61 6	1 8
	pruned, full	73	54 6	5 7

[a] Fitness is defined here as the number of pattern pairs in the best candidate solution that are present in the pattern pair table in the data set The "correct" fitness is the fitness of the "correct" solution

In the original data sets of Table I, we can observe the effects of missing and incorrect information

RESULTS

In Table III, the results of a series of five runs for the data sets are gathered As can be seen, in the original data sets, in almost all cases fitnesses are found that are higher than the fitness of the correct solution This indicates that a lot of local optima exist and that the optimum we are looking for will not even be the highest one In the completed data sets, of course, the correct solution has the highest possible fitness, and the previous behavior is not observed The column containing the standard deviations is also very interesting In three or four cases, markedly high standard deviations are found when the five GA runs are compared E-L30 with global and completed settings, BPTI with pruned and completed settings, and both incomplete and completed pruned settings of Tendamistat The BPTI case is the result of one run where the GA got stuck in a local optimum The standard deviation of the other four runs is much smaller In the other cases, rather different fitnesses are found, and it appears that the GA has difficulties in finding the correct path to the solution In the E-L30 case this may be caused by the large number of FN errors and, in the Tendamistat cases, by the large number of superfluous patterns that have to be mapped to the sequence of 74 amino acids Using the Tendamistat data sets that have not been pruned, high fitnesses can be obtained because of the enormous amount of possible pattern pairs in the input sets Thus, the number of local optima that are almost as high as the global optimum is very large We will see shortly that this hypothesis is confirmed by the low number of actually assigned patterns in Tendamistat

Results of the Original Data Sets. The original data sets in Table I present the most difficult cases since they contain the highest number of missing pattern pairs along with a large amount of superfluous pattern pairs In Tables IVand V the definite and possible assignments, respectively, for these data sets are tabulated for the different evaluation functions In all cases, the percentage of possible assignments that is correct is significantly smaller than the percentage of definite assignments This is as expected, but in most cases the additional number of correct assignments is so small that there

Table IV. Definite Assignments for the Data Sets from Table I with Different Evaluation Functions[a]

data set	bonus = 0	bonus = 0.5	bonus = 1	bonus = 2
E-L30, optimal	9/16	10/16	4/11	10/18
E-L30, global	7/15	5/15	6/15	6/20
BPTI, optimal	18/20	23/28	18/23	18/22
BPTI, global	12/17	17/23	14/16	13/22
Tendami, optimal	6/19	5/18	4/10	3/8
Tendami, global	3/10	3/9	4/10	4/10
total	55/97	63/109	50/85	54/100

[a] The first number of each entry is the number of correct assignments; the second number, the total number of assignments. Bonus is the number that is added to the fitness of a solution in the case of two successive correct pattern pairs. The case where bonus = 1 yields the highest percentage of correct assignments (59%) as well as the smallest number of incorrect assignments

Table V. Possible Assignments for the Data Sets from Table I with Different Evaluation Functions[a]

data set	bonus = 0	bonus = 0.5	bonus = 1	bonus = 2
F-L30, optimal	13/40	12/47	8/42	14/35
E-L30, global	13/38	9/39	7/31	8/43
BPTI, optimal	26/47	24/44	29/44	30/50
BPTI, global	14/40	28/50	18/39	15/41
Tendami, optimal	16/58	15/57	12/48	10/50
Tendami, global	9/47	9/44	9/53	8/37
total	91/270	97/281	83/257	85/246

[a] In this case, the highest performance is obtained where bonus = 2.

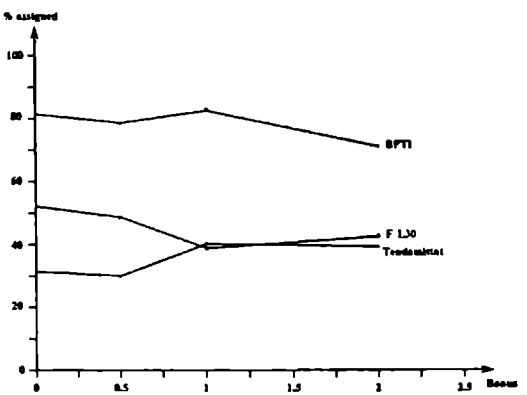

Figure 4. Percentage of correct definite assignments for the three proteins, using different values for the bonus parameter. These results are the mean over the optimal and global data sets (see text).

appears to be no benefit at all in bookkeeping the possible assignments. This will be consistently the case in all subsequent results, so for the sake of brevity we will not give the results of possible assignments further.

In Figure 4 the percentages of correct assignments are gathered for each protein, where the results using both the optimal and global data sets have been combined. It is clear that the GA performs best in the case of BPTI. In this data set, relatively few FN errors are present, and the load of some extra FP errors, compared with E-L30, does not seem to pose a problem. Here, roughly 80% of all assignments are correct. In E-L30, more patterns are incorrectly assigned because of errors in the input set. Tendami represents a different case because of the large amount of superfluous patterns and pattern pairs and the greater length of the amino acid chain. However, the difference between the global and optimal settings is small, especially in the possible assignments. It may be concluded

Table VI. Definite Assignments for the Pruned Data Sets from Table II with Different Evaluation Functions

data set	bonus = 0	bonus = 0.5	bonus = 1	bonus = 2
E-L30	6/26	18/29	18/28	16/23
BPTI	37/44	35/42	37/39	38/44
Tendami	29/35	26/34	22/32	14/23
total	72/105	79/105	77/99	68/90

Table VII. Definite Assignments for the Completed Data Sets from Table II with Different Evaluation Functions

data set	settings	bonus = 0	bonus = 0.5	bonus = 1	bonus = 2
E-L30	optimal	11/19	14/16	17/23	17/19
E-L30	global	25/32	27/29	24/31	19/20
E-L30	pruned	55/56	55/58	57/57	55/58
BPTI	optimal	18/24	36/39	23/27	33/38
BPTI	global	21/28	20/22	14/21	16/20
BPTI	pruned	49/52	51/51	51/53	52/54
Tendami	optimal	6/17	5/9	5/12	4/18
Tendami	global	5/18	9/19	6/11	8/16
Tendami	pruned	35/43	35/39	38/41	37/43

that both data sets are actually too indetermined to let the GA find a reasonable solution. The small numbers of definite assignments reflects the observed behavior that there are many ways in which a trial solution with a high fitness can be formed. Therefore, patterns are rarely assigned to the same place in two or three runs.

Results of the Constructed Data Sets. The constructed data sets from Table II present different situations to the genetic algorithm. In the manually pruned sets, much less superfluous pattern combinations are present, and in the full sets, all pattern combinations of the correct solution are present in the data set. Thus, with these sets, the sensitivity of the genetic algorithm with respect to false negative and false positive errors in the input can be investigated.

Comparison of the results of the pruned sets with the results of the optimal sets in Table IV gives an indication of the influence of FP errors on the eventual outcome. The results of the pruned sets are given in Table VI. Pruning clearly yields a significant performance improvement, as expected. Not only can the correct path to the global optimum be found easier, but also several other local optima are removed. The data sets where the missing pattern pairs are completed can be used to further assess the ability of the GA to find the global optimum, which in these cases comprises the correct solution. These results are gathered in Table VII.

DISCUSSION OF RESULTS

Original Data Sets. First of all, the results of the original data sets can be compared with each other (see Table IV). It is clear that the BPTI data sets give the best results. Despite significant numbers of FN, as well as FP errors in the input files, a reasonable assignment rate is achieved. In the case of E-L30, the number of correct assignments is much smaller, presumably because of the large number of FN errors; more than 50% of the correct pattern combinations is absent from the input files. The Tendamistat data sets present another difficulty: the number of possible pattern combinations is so large that a large number of sequences can be constructed that have an equal or even higher fitness than the correct sequence. In this case, many local optima of a very different nature can be found that have high fitnesses; and most of them have higher fitnesses than the correct solution (see Table III). This situation is adequately reflected by the very small number of assignments, of which, in many cases, only the

proline assignments are correct These are treated as dummies in the input data because HIPS does not assign proline patterns [12]

Pruned Data Sets. Second, the results of the pruned data sets may be used to assess the performance of the GA in the case of only a limited number of FP errors (compare the optimal data sets in Table IV with those in Table VI) In almost all cases, the number of assignments as well as the percentage of correct assignments increases significantly The difference is most markedly present in the case of Tendamistat, where the original data sets posed for the GA too big a problem, with the pruned Tendamistat data set approximately one-third of all positions is correctly assigned Also the results on the E-L30 and BPTI data sets show a significant improvement The interesting thing to note is that the largest pruned data set, Tendami, contains more than half the number of pattern combinations than the smallest original data set, and besides that a larger surplus of patterns for the available positions The difference in difficulties presented by these data sets thus appears to be rather small This notwithstanding, results of the pruned Tendamistat data set are significantly better than the results of the original E-L30 set Almost 40% of the positions have been assigned correctly in the pruned Tendamistat case, whereas in the original data set of E-L30, the best performance consists of a correct assignment of 17% of the positions However, the percentages of assignments that are correct differ only a little in these cases, it appears that a large number of FP errors predominantly prevents patterns from being assigned to a specific position From this, a tentative conclusion may be drawn that there is some cutoff value for the number of FP errors above which the GA will not be able to yield useful results On the basis of these data, one could estimate that the cutoff value would lie somewhere around 10–20 times the number of positions in the sequence If more FP errors are present, no reliable assignments are to be expected

Completed Data Sets. Third, the completed data sets may be compared with the incomplete sets to see how much the performance of the GA is hampered by FN errors Of course, a completely correct interpretation is almost impossible in the presence of such errors, but, as already said, it is already a significant aid in the spectrum interpretation if large parts of the sequence have been assigned As can be seen in Table VII, the effect of removing the FN errors is largest in the pruned data sets, where the FP errors do not play a predominant role and the number of assignments already is significant In the pruned E-L30 and BPTI cases, 85–100% of the positions is assigned the correct pattern In the Tendamistat data set, this figure lies around 50%, indicating that the number of approximately 450 FP errors is still too big to achieve a complete assignment In the original data sets, not so much of an improvement in the number of assignments has been reached, but more of an improvement in the rate of correct assignments However, the Tendamistat data sets still are too difficult for the GA to find the global optimum, that now constitutes the real solution

Evaluation Function. Finally, some comments on the evaluation function may be made The representation of a candidate solution and its evaluation criterion should not only represent the merits of that solution but should also enable the crossover operator to combine useful parts of different solutions so that an even better one is obtained This implies that the so-called "fitness landscape", that is effectively sampled by the GA, should have a more or less smooth surface If it is completely flat with one sharp spike, containing the solution, it is extremely unlikely that search methods will find the optimum, given solution spaces of the current magnitude On the other extreme, fitness landscapes with a very ragged surface without any apparent coherence will also give bad and unreliable results In such a case, the GA will not perform much better than a random search In general, it is best to leave the evaluation function as simple as possible, since complicated evaluation functions often have a tendency to roughen the fitness surface For instance, the first evaluation function that was tried consisted of a part in which probabilities of patterns belonging to certain types of amino acids were combined with the presence or absence of sequential cross-peaks Although the fitness gave a very good picture of what constituted a good solution, the results with the GA were very poor

The current evaluation function, however, is much simpler The "correct" solution has the highest fitness, but there are a lot of partially correct solutions that approach this fitness This is essential for a successful operation of the GA However, it was thought worthwhile to try the enhancement of the evaluation function by rewarding multiple vicinal pattern combinations that were present in the input set It can be seen, however, that the addition of a bonus in such a case does not seem to have much influence Although in some cases better results may be obtained, especially when averaging over the three proteins, the general trend is not very convincing, and further performance improvements are more likely to appear if the quality of the input sets is improved A further improvement could be the inclusion of more GA runs to determine whether or not a pattern is definitely assigned to a specific position

CONCLUSION

In this paper, the possibilities of using genetic algorithms in the sequential assignment of NMR protein spectra have been assessed It can be concluded that, provided the data sets are of sufficient quality, good results can be obtained The amount of errors that is permitted in the data sets has been investigated Useful results (one third of the amino acid positions assigned correctly) may still be obtained if the number of pattern combinations exceeds the necessary number of amino acid combinations with an order of magnitude These FP errors mainly affect the number of assignments that is made by the GA, and not the rate of correct assignments Missing information (FN errors) mainly affects the quality of the assignments, as may be expected If a pattern combination is absent from the input data set, the GA will try to fit in a false combination that will be rewarded in the evaluation phase Data sets with up to 50% missing pattern combinations have been evaluated, with the result of 10–30% of the positions correctly assigned, depending on the number of FP errors If the number of FP errors is small enough, and the number of FN errors is zero, performances of nearly 100% may be obtained

The above results indicate that the GA is a promising technique to be used in the automation of the spectrum interpretation process Especially in cases where an amount of errors is present that would prohibit other techniques from producing any results at all, the GA may still be able to assign a part of the sequence correctly The quality of the solutions presented by the GA may be assessed by investigating the fitnesses of the solutions as well as the number of assignments that can be made from the runs Because of the large robustness against both FP and FN errors, it is expected that this technique can be coupled effectively to expert systems or

other programs, where uncertainties may lead to a large number of possibilities and incorrect conclusions. However, it must be borne in mind that the spectra from which the data sets were derived have been simulated, albeit as realistically as possible. Results with real spectra must be obtained to adequately assess the power of the method and to be able to compare it with other methods.

REFERENCES AND NOTES

(1) Catasti, P ; Carrara, E.; Nicolini, C Pepto an expert system for automatic peak assignment of two-dimensional nuclear magnetic resonance spectra of proteins *J Comput. Chem* **1990**, *11*, 805–818.

(2) Ciesiar, C., Clore, G. M.; Gronenborn, A. M. Computer-aided sequential assignment of protein ^{1}H NMR spectra. *J. Magn. Reson.* **1988**, *80*, 119–127.

(3) Eads, C D ; Kunz, I. D Programs for computer-assisted sequential assignment of proteins. *J. Magn Reson.* **1989**, *82*, 467–482.

(4) Kleywegt, G. J , Lamerichs, R M J. N.; Boelens, R , Kaptein, R. Toward automatic assignment of protein ^{1}H NMR spectra. *J. Magn Reson.* **1989**, *85*, 186–197.

(5) van de Ven, F. J. M PROSPECT, a program for automated interpretation of 2D NMR spectra. *J. Magn. Reson* **1990**, *86*, 633–644.

(6) Yu, C.; Hwang, J.-F ; Chen, T.-B.; Soo, V -W. RUBIDIUM, a program for computer-aided assignment of two-dimensional NMR spectra of polypeptides. *J Chem Inf. Comput. Sci.* **1992**, *32*, 183–187.

(7) Wüthrich, K. *NMR of proteins and nucleic acids*; Wiley: New York, 1986.

(8) Goldberg, D. E *Genetic algorithms in search, optimization and machine learning*; Addison-Wesley· Reading, MA, 1989.

(9) Davis, L., Ed *Handbook of Genetic Algorithms*; Van Nostrand Reinhold. New York, 1991

(10) Lucasius, C B., Kateman, G. Understanding and using genetic algorithms Part 1: Concepts, properties and context. *Chemometrics and Intelligent Laboratory Systems*, 1993, in press.

(11) Lucasius, C. B., Kateman, G Understanding and using genetic algorithms. Part 2. Representation, configuration and hybridization. *Chemometric and Intelligent Laboratory Systems*; 1993, submitted for publication.

(12) Wehrens, R ; Lucasius, C.; Buydens, L ; Kateman, G. HIPS, a hybrid self-adapting expert system for NMR spectrum interpretation using genetic algorithms. *Anal Chim. Acta*, in press

(13) Lucasius, C B , Kateman, G Towards solving subset selection problems with the aid of the genetic algorithm In *Proceedings of the 2nd Workshop on Parallel Problem Solving from Nature*, Brussels; Männer, R., Manderick, B , Ed ; Elsevier· Amsterdam, 1992, p 239.

(14) Lucasius, C. B.; Kateman, G. GATES. Genetic Algorithm Toolbox for Evolutionary Search, Software Library in ANSI C, Laboratory for Analytical Chemistry, Catholic University, Nijmegen, 1991

CHAPTER 12

On k-Medoid Clustering of Large Data Sets with the Aid of a Genetic Algorithm: Background, Feasibility and Comparison

Contents

On k-Medoid Clustering of Large Data Sets with the Aid of a Genetic Algorithm: Background, Feasibility and Comparison

Abstract

A novel approach to the problem of k-medoid clustering of large data sets is presented, using a genetic algorithm Genetic algorithms comprise a family of optimization methods based loosely upon principles of natural evolution They have proven to be especially suited to tackle complex, large-scale optimization problems efficiently, including a rapidly growing variety of problems of practical utility. The pilot study presented in this paper, lays emphasis on the feasibility of GCA – our genetic algorithm for k-medoid clustering of large datasets – and provides some background information to elucidate differences with traditional approaches The experimental part of this study is done on the basis of artificial data sets and includes a comparison with CLARA – another approach to k-medoid clustering of large datasets, introduced recently. Results indicate that GCA accomplishes a better sampling of the combinatorial search space

1 Introduction

Cluster analysis comprises a widely applied science [7, 20, 28] within the rapidly growing area known as exploratory data analysis – a denominator for methods which explore the structure of data that does not require the assumptions common to many statistical methods. Numerous applications reported in the literature concern analytical-chemical problems [24, 61, 48, 23, 57, 50, 64].

In brief, a cluster analysis is concerned with the problem of finding k groups – or partitions, or similarity classes, or indeed clusters – of objects in a metric data matrix according to some plausible (dis)similarity criterion, thereby revealing hidden, *a priori* unknown, structure. The objects, or patterns, are the rows in the data matrix; each pattern among the K patterns ($k \leq K$), lists L numerical values for a given set of variables that describe some system. A particular partitioning must fulfil the condition that each partition contains at least one object, and that each object belongs to exactly one partition.

Exploratory data analysis methods fit in the broader class of data reduction methods – methods which aim to explicit hidden structure in a $K \times L$ data matrix. Data reduction in cluster analysis is effectuated when a pattern is selected from each of the k clusters and considered as a representative of that cluster. From this viewpoint, a clustering problem can also be seen as a subset selection problem: the selection of a subset of k representative patterns from the source set of K patterns. It should be realized that pattern selection is but one way to accomplish data reduction. In other approaches to data reduction, one reduces the L variables to l variables (dimensionality reduction), using some criterion in order to minimize the loss of information. Variable reduction can be accomplished either by the selection of a subset of l variables from the source set of L variables, or by the creation of l new (*ad hoc*, or latent) variables such as in principal components analysis (linear, Karhunen-Loeve projection) or in non-linear mapping [50] in generalized form. Genetic algorithms appear to be competitive in both variable reduction [58, 41, 37, 36, 39, 40] and pattern reduction [38].

Method validation criteria In this paper, generally important criteria for the validation of computational methods are assumed to be:

- The quality of the end solution found by the method:
- The amount of computation time required by the method,
  here, "computation time" is defined as the time the method uses in order to terminate

according to some criterion other than an imposed maximum time that the user is prepared to wait in practice, e.g. according to a convergence criterion or a criterion based on a predefined acceptable quality of the estimated solution; in contrast, "running time" is defined as the time that the method is allowed to run in practice, never exceeding a predetermined maximum waiting time;

- The amount of computer memory required by the method;
- The ease of developing and implementing the method;
- The ease of using the method.

In the course of this paper, these criteria are regularly addressed in order to validate the methods of interest. For the moment, it is important to realize that these criteria can be mutually conflicting in practice. For instance, it may be difficult to maximize the quality of the end solution and at the same time minimize the amount of computation time. Indeed, it turns out in practice that simultaneous optimization of the criteria often forces one to make compromises.

2 Combinatorial optimization approach to clustering

In this section, we examine clustering as a combinatorial optimization problem and motivate the choice for a so-called partitional clustering strategy when large data sets are concerned. A genetic algorithm is proposed as a strategy that can be brought to bear competitively on partitional clustering.

2.1 Measures of dissimilarity for optimality criteria

The collection of all legal clusterings, or partitionings, for a particular clustering problem is hereafter called the search space. In searching for the best partitioning in this space, an optimality criterion – or evaluation criterion, or objective function – is applied in order to rate any candidate partitioning according to a plausible figure of merit (utility, quality). Here, "plausible" indicates the intuition that a particular clustering is better when the average dissimilarity between intra-cluster objects is smaller, while the average dissimilarity between inter-cluster objects is larger. Many ways seem acceptable to formalize this notion into an optimality criterion. An example is an optimality criterion based on the so-called k-medoid model, which comes up in several sections below.

Pairwise inter-object dissimilarities, or distances, are denoted as d_{ij}, where i and j are unique indices attached to the objects ($1 \leq \{i,j\} \leq K$). The distances can be calculated according to different measures, e.g. [20] the Minkowski distance, Mahalanobis distance, and $1 - |r_{ij}|$ (where r_{ij} denotes the correlation coefficient) The distances are elements in a square, $K \times K$, distance matrix, $\mathbf{D}$, which is mathematically derived from the $K \times L$ data matrix (i.e. the data set), $\mathbf{X}$. Owing to the properties $d_{ij} = d_{ji}$ (symmetry) and $d_{ii} = 0$, only the $K(K-1)/2$ d_{ij} values for $i < j$ – comprising the upper triangle of $\mathbf{D}$ (diagonal exclusive) – need to be stored in computer memory or calculated in run-time as they are needed. Incidentally, for larger L values it becomes more attractive to store the d_{ij} values in memory, because the time of their calculation is proportional to L.

The second order Minkowski distance, also known as the Euclidean distance, is popular and therefore adopted in our study. Moreover, it is hereafter implicitly assumed that $\mathbf{D}$ is calculated from an autoscaled (that is, z-transformed) $\mathbf{X}$; in this way, equal importance is given to the L variables.

2.2 Sampling the search space

Clustering techniques which have apparently enjoyed most widespread application up to now, fall within the categories *hierarchical clustering* and *partitional clustering* [48, 50, 20, 28].

In order to motivate our choice for partitional clustering, we briefly pass in review the sampling properties of hierarchical clustering first. The main shortcoming of hierarchical clustering is shown to be related to a "greedy" way in which the search space is sampled. General consensus exists that partitional clustering methods amount to a much better sampling scheme, in general. That this optimism is not entirely justified, follows in

due course when we discuss a recently developed, highly praised partitional clustering method and show that it too is in fact based on a greedy sampling scheme which, although certainly quite different from an hierarchical sampling scheme, features some striking parallels with the hierarchical scheme in a more subtle sense.

2.2.1 Properties of hierarchical clustering

Hierarchical clustering is traditionally most applied, for its speed and its ease of implementation and use. A sequence of nested partitionings is generated incrementally, such that at the bottom level each object forms a separate cluster, or singleton ($k = K$), and at the top level all objects form one single cluster ($k = 1$); hence, all values of k are dealt with in the same run. One of the $K-2$ intermediate levels represents the solution of the problem, and is chosen at the discretion of the user (through visual inspection of a graphical representation of the sequence, called a dendrogram). The procedure is either started at the bottom level, merging one object with a cluster at a time (agglomerative strategy), or at the top level, splitting one object from a cluster at a time (divisive strategy). In either case, each step is performed in a greedy way in that the merging/splitting object is selected deterministically (i.e. not probabilistically) according to the greatest relative gain in overall similarity, and the choice is irreversible (i.e. can not be canceled at a later stage if that would then increase overall similarity).

The most important shortcoming of hierarchical clustering is the poor sampling of the search space: due to greedy nature of the search heuristic, only a tiny fraction of the search space is sampled. Adding to that the presence of sub-optimal partitionings, it turns out that this fraction does not necessarily contain the best partitioning or even an acceptable partitioning. More quantitatively, the number of partitions that are evaluated is:

$$(K-1)+(K-2)+\cdots+2 = (K+1)(K-2)/2$$

For $K = 19$, this is 170. Indeed, the search space is much larger, considering that the number of legal partitionings for given K and k is:

$$P_k^K = \frac{1}{k!}\sum_{i=1}^{k}(-1)^{k-i}\frac{k!}{i!(k-i)!}i^K \approx \frac{k^K}{k!} \quad (1)$$

For $K = 19$ and $k = 4$, say, this is 11,259,666,000 – merely the size of a subspace in the entire search space defined by a fixed K. (The size of the entire search space is obtained by summing for $k = 1, \cdots, K$.) — When k is kept fixed, the size of the search space grows exponentially with K (reaching astronomical proportions quickly), whereas the portion sampled during the search grows only as a quadratic function of K; we say that hierarchical clustering has a *time complexity* of $O(K^2)$, or order 2 in K. Thus, with increasing K, the disparity between the size of the search space and that of the portion sampled grows rapidly. It follows from practice that for large data sets – more precisely, typically for $K > 100$ – the solutions found by hierarchical clustering are no longer acceptable, as a rule; the comparatively short convergence time is then meaningless, of course.

2.2.2 Properties of partitional clustering

There is a general consensus among practitioners that when it comes to clustering of large data sets, a non-hierarchical, or partitional, clustering method is imperative. By definition, such a method employs a search heuristic which performs steps in the search space that are not confined to irreversible merges/splits of objects with/from clusters, thus allowing a better sampling of the search space.

Partitional clustering problems – or rather partitioning problems in general – are provably $\mathcal{NP}$-complete [47]. Formally, this means that it is impossible, or at least extremely difficult, to obtain a search heuristic that can find the true solution of the problem in polynomial time [45]. Intuitively, a practically appropriate search heuristic must be "intelligent", i.e. neither too "weak" (e.g. not exhaustively scanning the search space) nor too "strong" (i.e. not too greedy); such search heuristics are also called "moderate" [45] – reasonably efficient and reasonably reliable (robust).

Countless partitional clustering methods are conceivable. Approaches which guarantee that the best partitioning will be found – e.g. enumerative search – are not practically feasible, as the entire search space must be scanned exhaustively; this is an astronomical enterprise for problems of interest ($K > 100$). On the other hand, for practically feasible approaches, among which some are briefly passed in review in the next section, it is not guar-

anteed that the best partitioning will be found, albeit near-optimal or acceptable partitionings can be obtained, in general.

At this juncture, we emphasize that in the remainder of this paper its is implicitly assumed that $K > 100$ and $k \ll K$ (typically $k < K/10$), unless explicitly stated otherwise.

2.3 Approaches to partitional clustering

Among putatively intelligent methods for partitional clustering are the following.

The branch and bound method [34] has been applied to partitional clustering by Koontz *et al.* [31] and later by Massart *et al.* [49]. However, the method has been criticized as computationally too expensive for the large data sets frequently encountered in practice. Moreover, the method is based on assumptions which are often not realistic in practice, *viz.*: the optimality criterion must satisfy monotonicity (a subset should not be better than any larger set that contains it) and a lowerbound on the optimum (maximum) must be known. Also, the development and implementation of the method is sometimes criticized as being error-prone due to the complex bookkeeping explicitly needed.

Simulated annealing [29, 33, 1] has been applied to partitional clustering by Klein and Dubes [30]. This method is based on probabilistic (i.e. nondeterministic) search heuristics that are relatively easy to develop and implement. An advantage of probabilistic search heuristics, in general, is that end solutions do not depend strongly on the initial estimate. Good results have been obtained in various applications. However, the space of "cooling" schedules is huge and complex, i.e. to find an acceptable configuration of the method is normally troublesome.

Recently, artificial neural networks have also found promising application to partitional clustering [25, 51, 59]. In this methodology too, the configuration space is huge and complex, and finding an acceptable configuration is normally troublesome.

Tabu search [10, 11, 54] has been proposed for large-scale combinatorial problem solving, inter alia, and can thus be brought to bear on partitional clustering. (An application of tabu search involving hierarchical clustering is described in [15].)

A method apparently tailored to partitional clustering – CLARA, from Clustering LARge Applications – was recently introduced by Kaufman and Rousseeuw, and later also applied by Hopke [27, 26, 19, 18, 28]. Based on these pilot studies, CLARA has been praised for its efficiency and robustness attributed to a reportedly unique, intelligent statistical resampling heuristic. CLARA is founded upon the aforementioned k-medoid model, and is used for the comparative part of our study.

2.4 Motivation for a genetic algorithm

Genetic algorithms [16, 12, 6, 4, 52, 17, 32, 45, 46] have been applied competitively to various complex, large-scale optimization problems of practical importance [12, 13, 14, 56, 2, 6] (including several problems in analytical chemistry, referenced in [45, 43]), and therefore naturally seem a profitable choice to tackle partitional clustering problems. Genetic algorithms for partitioning problems in general, have only recently attracted the attention of some scientists [3, 35, 21, 9, 8]. (Historically, genetic algorithms were first predominantly applied to numerical parameter estimation problems – later also substantially to sequencing problems, e.g. scheduling problems. At this point in time, one can only speculate on the impact that genetic algorithms are going to have on partitioning problems.)

One of the, up to now few, studies involving genetic algorithms for partitional clustering specifically, is due to Bhuyan *et al.* [3]. They conducted a comparison with a greedy partitional clustering technique, and showed that the genetic algorithm performs better. However, they did not use large data sets ($K < 60$). Our study is different in that we challenge large data sets ($K = 1000$) and perform a comparison with another partitional clustering technique (CLARA).

The power (efficiency and robustness) of genetic algorithms can be intuitively appreciated by considering one of their distinguishing characteristics, namely: not one candidate solution, but rather a collection of candidate solutions, in encoded form, is iteratively modified by the search heuristics. In this way, the search is multi-directional; that is, multiple searches are carried out in a single run. Importantly, as these searches interact · i.e. ex-

change information to enhance mutual guidance – the overall search can become very efficient. The collection, or *population*, of encoded candidate solutions undergoes a simulated evolution process reminiscent of natural evolution according to Darwin; that is, at each time step, or *generation*, in which the population is updated, the relatively well ranking solutions reproduce and create offspring, while the relatively low ranking solutions are repelled from the population. More light on the mechanics of genetic algorithms is shed in Section 3.3.1 below.

3 Theory

In this section, we give a summary of CLARA as it has been described in the literature, and introduce a genetic algorithm for partitional clustering – named GCA, from Genetic Clustering Algorithm. CLARA and GCA employ the same objective function, based on the k-medoid model.

3.1 The k-medoid framework

A clustering heuristic that is based on the k-medoid model [49, 27, 26, 19, 18, 28] evaluates a set of k selected objects considered representative for the k clusters to be found within the source set of K objects. Given the set of representative objects, the remaining objects are assigned to the nearest representative object (Figure 1), using the chosen distance measure (Euclidean distance in our case). The underlying philosophy of this deterministic *object assignment* procedure is that a better set of clusters is obtained when the k representative objects are more centrally located in the cluster they define. For this reason, the (supposedly) optimal representative objects are called medoids, and a suitable objective function to be minimized is the sum, D_k, of the distances of the respective medoids to all the other objects of the same cluster:

$$D_k = \sum_{\kappa=1}^{k} \sum_{\substack{i \text{ assigned} \\ \text{to medoid } \kappa}} d_{\kappa i} \qquad (2)$$

where the $d_{\kappa i}$ values are retrieved from the distance matrix $\mathbf{D}$. Alternatively, the average distance, $\delta_k = D_k/(K-k)$, can be minimized, but the calculations are then numerically less accurate [28].

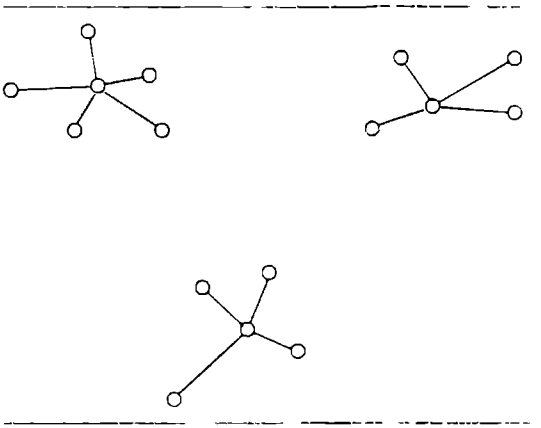

Figure 1: Illustration of the k-medoid model: object assignments to medoids.

Importantly, due to the object assignment implied by the k-medoid model, the target k-ary (k-way) partitioning problem is, in essence, reduced to a binary (2-way) partitioning problem, or subset selection problem; from *this* perspective, the first partition is the subset of the k medoids to be selected, and the second partition is the thereby implied complementary subset of the $K-k$ non-medoid objects. Accordingly, the number of possible k-medoid subsets comprising the search space is given by:

$$S_k^K = \frac{K!}{k!(K-k)!} \qquad (3)$$

which, although sizably smaller than P_k^K (Equation 1), is in practice still beyond exhaustive search when large data sets are concerned. For $k \ll K$, S_k^K grows explosively with k. (By "explosive" we mean "faster than quadratically – in many cases even faster than a higher-order polynomial".)

Other important characteristics of the k-medoid model are: (1) the implicit assumption of hyperspherical clusters in the L-dimensional data space; (2) the objective function (D_k) exhibits bias for large k values, as can be readily appreciated by considering the extreme case of singleton clusters ($k = K$, resulting in $D_k = 0$).

Due to the bias in the objective function, a clustering method based on the k-medoid model is normally configured such that k remains constant during the search; in that case, therefore, we speak of a fixed-size subset selection problem [46, 43]. (Ideally, one would prefer to approach

k-medoid clustering as an unknown-size subset selection problem, as that would allow one to implicitly obtain the best k in a single run.) In practice, runs are often repeated for different values of k, and the best k is found according to an off-line cluster validation criterion applied to the results; one such a criterion is based on the recently introduced silhouette analysis [55, 19, 18, 28], which however lies outside the scope of this paper.

Any clustering method based on the k-medoid model is said to belong to the class of k-medoid clustering methods, although the most popular method therein is often referred to as "the" k-medoid method. Recently, the name PAM – from Partitioning Around Medoids – was coined for the k-medoid method. Incidentally, relative to the so-called k-means method [28], developed earlier, PAM is often praised for its: (1) robustness; (2) good characterization of all clusters that are not too elongated; (3) good detection of outliers.

PAM consists of two procedures: a *building* procedure followed by a *swapping* procedure, each based on a different incremental stepping heuristic. In general, incremental stepping heuristics (including the abovementioned hierarchical clustering methods) have a time complexity that is quadratic in the dimensionality of the system under consideration. Indeed, both procedures in PAM can be estimated to have approximately time complexity $O(K^2)$ for fixed k; also, the time complexity is approximately $O(k^2)$ for fixed K. (In making these estimates, it is assumed that the computation time of PAM is predominantly determined by the summation of distances to calculate D_k; this is reasonable, because the burden of other computations – mostly distance retrievals from $\mathbf{D}$ stored in memory, and object assignments – is, in comparison, negligible.)

For a detailed description of PAM, the reader is referred to [28]; here, we suffice with the following concise outline.

In the building procedure, the first step concerns the selection of the object that represents the best medoid if $k = 1$; that is, D_1 is minimized – all objects are considered to belong to the same cluster. In each consecutive step, a new medoid is added to the set of medoids, namely the object which maximizes some predefined contribution (relative gain); the stepping goes on until all k medoids are selected. Although the contributions evaluated in each step may be regarded as the values delivered by an objective function other than D_k, the overall procedure is supposed to minimize D_k; that is, the building procedure aims to minimize D_k indirectly.

Using the set of k medoids thus obtained, the swapping procedure attempts to further improve this set by successively considering all pairs of objects for which one object is a medoid and the other is not. In each pair, the roles of medoid and non-medoid are interchanged as part of a strategy to calculate some predefined contribution, and in each time-step the exchange which yields the largest contribution is accepted. The procedure terminates when there are no more pairs for which an exchange leads to improvement. Note that the swapping procedure, too, aims to minimize D_k indirectly by maximizing contributions in the respective steps performed.

It is to be ascribed to PAM's quadratic time complexity in K that computation times become unacceptably long in practice when large data sets are used. One way to potentially circumvent this problem is to estimate the best clustering of all objects from *samples* of objects passed to PAM. Such an approach – statistical resampling – is embodied in CLARA, discussed next.

3.2 CLARA

In CLARA [28], PAM is called M times by an embedding, or major, iteration cycle, where M is a control parameter; in this vein, PAM may be considered the embedded, or minor, iteration cycle of CLARA. In each major iteration, PAM performs a clustering of K' objects selected from the K objects in the entire data set. The value of K' ($< K$) is chosen according to the equation $K' = Ak + B$, where A and B are control parameters; the standard values for M, A, and B are listed in Section 4.3 below.

In the first major iteration, the sample of K' objects for PAM is selected randomly from the K objects. After PAM has completed clustering this sample, the remaining $K - K'$ objects are quickly assigned to the respective nearest k medoids; next, D_k (Equation 2) is calculated directly for all objects thus clustered. Subsequent major iterations follow a similar scheme, except that the sample of K' objects for PAM is now required to include the set of k medoids most recently found and accepted. Only if the new set of medoids found turns out to

be better (i.e. features a lower D_k) than the most recently accepted set, it will be accepted for use in the next major iteration; otherwise it is rejected and the most recently accepted set is reused in the next major iteration.

3.2.1 Computation time

As a consequence of CLARA's quadratic time complexity in k (assuming $K' = Ak + B$), any reasonable maximum allocated running time is reached quickly as k increases. This is probably an important reason why, to our knowledge, the few hitherto reported applications of CLARA typically feature $k < 10$. However, a more important reason why small k values are used, probably lies in the fact that the size of the search space grows explosively with increasing k (Equation 3); that is, with increasing k, the disparity between the size of the search space and that of the portion sampled grows rapidly. Ironically, a similar drawback was diagnosed in Section 2.2.1 above for hierarchical clustering. These arguments should warn us that CLARA is actually a greedy partitional clustering method, even though this has hitherto not been mentioned so explicitly in the literature. Therefore, it seems reasonable to suspect, at this juncture, that with increasing k, the search is increasingly likely to converge into sub-optima, especially when complex data sets are concerned. Whether this is indeed the case, can only be assessed empirically; this issue is addressed in the experimental part of our study.

3.3 GCA

In this section, we discuss a general flowchart of genetic algorithms, and indicate how it applies to GCA in particular. The flowchart comprises modules concerned with the exploration of the search space, as well as modules concerned with the exploitation of useful information collected along the way. The need for a balance between exploration and exploitation, and other general conditions for optimal performance, are emphasized. In general, the optimal choice of the exploration operators in a genetic algorithm depends on the type of domain under consideration, as will become evident in the course of the discussion.

As opposed to CLARA (or embedded PAM, actually), GCA calculates D_k directly in order to evaluate each candidate solution. This allows a better sampling of the search space.

3.3.1 Flowchart

Genetic algorithms comprise a family of probabilistic optimization methods based loosely upon principles of natural evolution. That is to say, they fulfil a widely accepted but not very strickt description, or generic flowchart [46, 43]:

> *See Chapter 2 (Section 3) in this thesis.*
> (In the chapter at hand, references made to other parts in this thesis are assumed to imply the there used terminology and conventions, unless explicitly overridden.)

3.3.2 Principal conditions for optimal performance

Two important conditions for efficient and efficacious search are [45, 46, 43]:

> *See Chapter 2 (Section 4.4) in this thesis.*

3.3.3 The exploration operators

For the comparison of GCA with CLARA to be meaningful, the modification operators (recombination and mutation) in GCA are based on the k-medoid model; that is, the operators modify strings in such a way that each new string still represents a legal set of k medoids selected from the source set of K objects.

Recombination. Initially, it was our intention to use the recombination operators that have recently become available for k-way partitioning problems in general [3, 35, 21, 9, 8]; for $k = 2$, then, our purpose would in principle be served – considering the two partitions in this case as, respectively, the subset of medoids and the thereby implied complementary subset of non-medoid objects. However, upon closer inspection these operators turn out to fall short seriously, since they regard finding k, as well as finding the sizes of the k partitions, a part of the search task. In this way, these operators have an unacceptable amount of overhead (redundancy) associated with them for

our purposes. This provided us with the incentive to design home-made recombination operators dedicated to fixed-size subset selection problems.

Our recombination operators for fixed-size subset selection problems are [42, 46, 44] **D_SX** (general-purpose subset recombination) and **D_MX** (mix subset recombination); the prepended "D" in the names serves to remind of the fact that direct (rather than binary) subset encoding is used [46, 43, 44], i.e. the strings that represent candidate subset are simply a concatenation of indices that directly and uniquely denote the selected elements. Unlike **D_MX**, **D_SX** can be used in additional modes, wherein it is aimed to preserve, respectively, the order and position of elements in the parent strings [42, 44, 62, 63]. These additional modes are, however, meaningless for k-medoid selection; that is, if **D_SX** is used, then only its regular mode (wherein the placement of elements is immaterial) would make sense for our purposes. Empirically, we established that **D_SX** in regular mode results in performance similar to **D_MX**. The latter is more memory efficient, and was therefore implemented in GCA.

In general, many recombination operators, including **D_MX**, when applied to two parent strings (P_1, P_2), produce two child strings (C_1, C_2):

$$P_1, P_2 \xrightarrow{\text{D\_MX}} C_1, C_2$$

The procedure for the swapping of string fractions that is part of any recombination operator, is applied with probability p_r – the recombination probability; otherwise, C_1 and C_2 simply become copies of P_1 and P_2, respectively. **D_MX** takes into account the constraints related to direct subset encoding, namely: any element that represents an object in the source set of objects, should occur at most once in the string of elements that represents a candidate subset. **D_MX** is defined as follows:

See Chapter 2 (Section 5.4.2) in this thesis.

Mutation. Besides the built-in mutation, a point mutation operator, **D_PM**, is independently applied to all strings in the population: one element in a child string C is selected randomly with a predetermined probability, $p_{m,\text{point}}$, and, upon success, replaced by a copy of an element indicated randomly in the complementary subset to produce C':

$$C \xrightarrow{\text{D\_PM}} C'$$

where, for instance, C = 3 6 8 and C' = 3 4 8. (Incidentally, **D_PM** is the equivalent of one iteration in **D_TM** – trade mutation, described in [42, 44].)

3.3.4 Computation time

In any genetic algorithm run, a population comprising N strings evolves for G generations. In many implementations, including GCA, by far most of the computational effort resides in the NG string evaluations. In GCA, the time of one such evaluation is proportional to the $K - k$ distances summed to obtain D_k. Thus, the computation time of GCA is proportional to $NG(K - k)$.

Optimal values for N and G depend on how one chooses to define "optimal", e.g. as minimal D_k (i.e. maximal quality of the end solution); or as some compromise between minimal NG (i.e. minimal computation time) and minimal D_k. Irrespective of the criterion is used, however, it is not straightforward how to theoretically derive optimal values for N and G. This may be ascribed to the mechanical complexity of genetic algorithms, in general, which obstructs reliable theoretical modeling of the search, thus making an empirical approach imperative. Intuitively, it is reasonable to expect that optimal values for N and G are such that their product, NG, increases with increasing problem dimensionality (defined by K and k).

For all practical purposes, the computation time of a particular, not necessarily optimally configured genetic algorithm may be taken equal to the convergence time, i.e. to the time after which D_k no longer decreases noticeably. This definition is convenient because convergence can easily be detected empirically (using some predefined convergence criterion).

Although the theoretical relation between the convergence time and the problem dimensionality is generally unknown, there exist theoretical arguments in favor of the statement that genetic algorithms can reach time complexities that are a small, possibly even fractional, power of the problem dimensionality – while (near-)optimal or acceptable solutions are still found. This attractive

property stems from the so-called *schema theorem*: the mathematical framework of genetic algorithms. It predicts that the search space is sampled exponentially in time if the principal conditions for optimal performance (as discussed in Section 3.3.2 above) apply [45]. More precisely, as population members – points in the search space – are explicitly evaluated, an exponentially growing portion of the search space is implicitly sampled; the implicit sampling is a consequence of the fact that schemata – or, basically, similarities between points in the search space – are taken advantage of during the search. Implicit exponential sampling of the search space distinguishes genetic algorithms from other search methods. It is widely believed that this property is responsible for the empirically established fact that a linear increase in the problem dimensionality – which normally leads to an exponential increase in the size of the search space – usually does not cause an exponential increase in the computation time: instead, a low-order polynomial increase is observed in most studied cases. (In the experimental part of our study, we aim at making a similar observation.)

4 Experimental

Here the computer programs used in this study are described, the comparison strategy followed is outlined, and the configuration of GCA is discussed in detail.

4.1 The computer programs

All computer programs were developed in the computer language C for portability and speed of execution. They were run on a SUN Sparc Station 2 (under UNIX). Using this hardware, for GCA the maximum allocated running time of a few hours was empirically found to correspond approximately with $NG < 100000$ (the number of string evaluations). This constraint was taken into account in empirically deriving optimal values for N and G, as described in Section 4.4 below.

CLARA. The computer program for CLARA was created from scratch, as described in [28]. For small values of k, it can be empirically established that the convergence time (computation time) of CLARA is well within a reasonable maximum allocated running time – e.g. within a few hours, say – employing abovementioned type of computer; this convergence does not automatically imply that the best- or even an acceptable solution is found. The statement that a few hours is a reasonable running time, is based on the assumption that other analytical activities can be carried out independently while computation is performed, or computation can be performed overnight; moreover, computation is nowadays fast and relatively inexpensive.

GCA. The computer program for GCA was created using a so-called genetic algorithm prototype in GATES – Genetic Algorithm Toolbox for Evolutionary Search – a library of domain-independent routines [43, 44]; by "domain-independent" routines are meant routines that can be used for other application domains as well. Among the prototypes available in GATES, the one dedicated to regular, fixed-size subset selection problems was chosen. Simply phrased, a prototype is a prefabricated, template genetic algorithm without an objective function. Accordingly, GCA was basically obtained by merely programming the objective function and linking it to the prototype.

In practice, the convergence time of a genetic algorithm often turns out to be larger than that of a more locally (greedily) searching method applied to the same problem. While this seems to contradict the theoretical claims of efficiency, it should be realized that, as evidenced by many practical applications, genetic algorithms tend to find solutions of considerably better quality. Empirically, we established that the computation time of GCA, as applied to the data sets used in this study, approximates a reasonable maximum running time – a few hours – employing abovementioned type of computer.

In both CLARA and GCA, **D**'s upper triangle, or hereafter simply **D** for short, is calculated from **X** and then stored in memory in a reduced form. The motivation for the reduction is that otherwise **D** occupies an unacceptable amount of memory for large K values of interest, as the amount needed grows quadratically with K.

In CLARA, the reduction in **D** is accomplished by taking advantage of the fact that in each of the M major iterations, PAM considers only a sample of ($K' < K$) objects. More precisely, in each major iteration, **D** is calculated only for the objects in the sample, and stored in memory for quick access

by PAM as the latter executes. A modification to CLARA that we believe is worth considering in the future, is the following. Before CLARA starts its major iterations, **D** is calculated from all K objects and then stored in memory in a compressed format, as elucidated shortly. In each iteration, **D** (compressed) is passed to PAM, along with a vector of K' integers that serve to index the objects in a particular sample. Using this vector of indices, compressed distances are quickly retrieved from **D** in PAM as they are needed.

Reduction in **D** through compression is implemented in GCA. The compression comes down to encoding real values as integers. In order to store a real number, e.g. a distance d_{ij} $(i < j)$, at least 64 bits are needed on many computers. We accomplish the compression of d_{ij} into a bitfield (bit row) of $\ell < 64$ bits by calculating the integer compressed distance, d^*_{ij}, for d_{ij} according to:

$$d^*_{ij} = \text{integral part of } (2^\ell - 1)\frac{d_{ij}}{d_{\max}} \qquad (4)$$

where $d_{\max}$ is the maximum d_{ij} in (uncompressed) **D** and ℓ is the chosen digitization resolution (in bits); in order to avoid an unacceptable truncation error, ℓ must not be too small. The bitfields of the respective d^*_{ij} values are concatenated contiguously to form a so-called compact bitstring, i.e. a bitstring in which all bits are actual computer bits (instead of, say, bytes confined to the values 0 and 1); d^*_{ij} is accessed as the h^{th} bitfield on the compact bitstring, where $h = j + (i-1)K - i(i+1)/2$ $(i < j)$; the technicalities of bitfield access on compact bitstrings are elucidated in [44]. In our study, we used $\ell = 10$ for all d^*_{ij} values; this amounts to a compression of nearly 85% relative to 64-bit real numbers.

Upon decompression, a digital real value – the decoded distance – is obtained:

$$d_{ij} = \frac{d^*_{ij}}{2^\ell - 1} d_{\max} \qquad (5)$$

The summation of d_{ij} values, then, yields D_k. However, since optimizing D^*_k – the sum of d^*_{ij} values – hereafter oftentimes referred to as "response" – is equivalent with optimizing D_k, runtime decompression is not necessary and has therefore been omitted from GCA. (The equivalence applies since the same ℓ is used for all d^*_{ij} values, thus conserving mutual importance among the L variables.)

GENERATE. The computer program GENERATE was developed for the creation of artificial data sets. The rationale for using artificial datasets is that the structure is *a priori* known, thus enabling a better validation of the methods used [53, 20]; this is important at the present stage wherein still little is known about the relative viability of genetic algorithms as applied to clustering problems. GENERATE complies with the k-medoid model in that it creates hyperspherical clusters according to a method described in [20]; furthermore, the clusters created by this method feature Gaussian distributed object densities, and we purposefully avoided overlap between the clusters.

Three data sets were created for our study, each comprising $K = 1000$ objects – in a two-dimensional data space $(L = 2)$ to allow graphical display – but different amounts of clusters: $k = 15$, $k = 25$ and $k = 50$; these (autoscaled) data sets are contained in files named `dat15.x`, `dat25.x`, and `dat50.x`, respectively, and are illustrated in Figure 2. (The extension `.x` reminds of the formal notation of the data matrix, **X**.)

4.2 Comparison strategy

A comparison between two methods is meaningful if both are optimally configured; when only one method is optimally configured and the other turns out to perform better nonetheless, then too a comparison is meaningful, although this can only be concluded afterwards, of course. Unfortunately, however, for many optimization methods used in practice, including CLARA and GCA, there is no standard methodology available to find an optimal working point in their configuration space. Therefore, in practice, one needs to accept the best configuration found for each method after evaluating a fair but limited amount of configurations, using the best available means. Since the comparison can not be made more reliable in practice, due reserve in drawing conclusions is in place.

4.3 Configuration of CLARA

The configuration of CLARA has been optimized earlier by its creators and the result was presented in the literature [28]. Since further optimization is beyond the purposes of this pilot study, we adopted this standard configuration: $M = 5$ (the

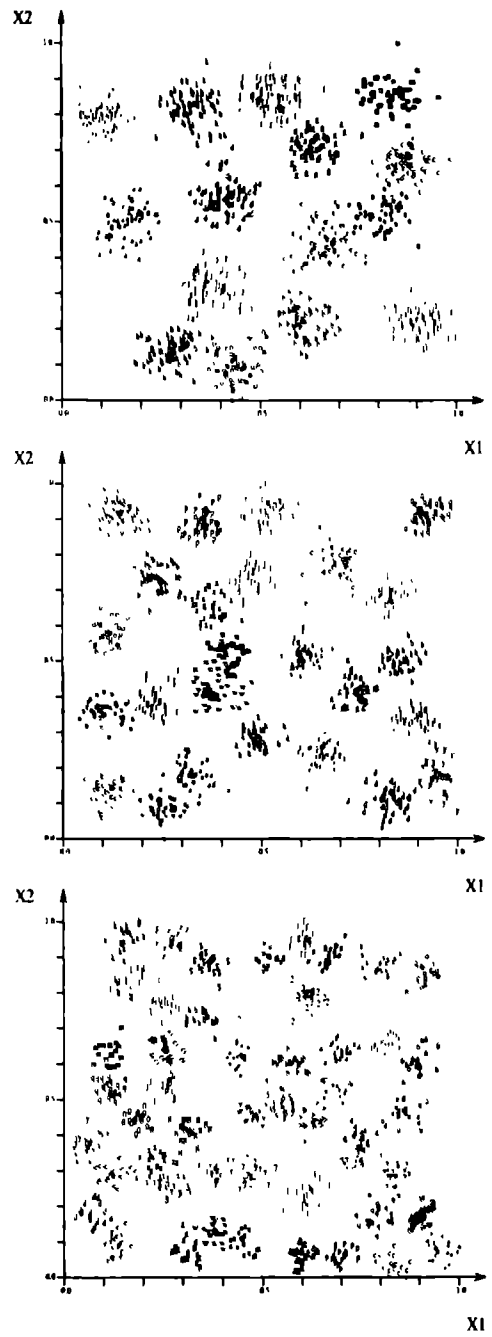

Figure 2: Artificial data sets for $K = 1000$, $L = 2$. Top: $k = 15$. Middle: $k = 25$. Bottom: $k = 50$. (Cluster membership is denoted by arbitrarily chosen characters.)

number of major iterations), $A = 2$ and $B = 40$ in $K' = Ak + B$ (the number of objects in the sample of objects passed to PAM).

4.4 Configuration of GCA

In general, the most successful strategy to optimize the configuration of a genetic algorithm turns out to be making educated estimates based on rules of thumb obtained through practice, i.e. on expert intuition [45, 46, 43]. The disadvantage of this approach, however, is that the expert involved can often not rationally explain the choices made.

Another approach that might be considered is a fractional factorial design, e.g. a Plackett-Burman design [22]. (A full factorial design, or grid search, would provide a guaranteed means to find the optimal configuration, but is not feasible within practically reasonable time due to the large amount of factors involved.) The disadvantage of fractional factorial designs is that they are based on strong assumptions, e.g. on the assumption that the factors involved are not correlated. Since such assumptions are not acceptable for a complex matter such as genetic configuration (where the factors are genetic control parameters, among which many are strongly correlated), it is important to practise great restraint in drawing conclusions from the results.

Our approach to the configuration of GCA is a combination of the abovementioned two approaches, as follows. Using the experience we have with the application of genetic algorithms, we selected a number of supposedly influential factors (control parameters) and made educated guesses at two more or less realistic levels for these; in this way, it is not likely that the space of configurations considered contains very bad configurations. These levels were used in a Plackett-Burman design to study the factor effects; the response measured to calculate these effects was the D_k^* of the best string in the population of the generation of interest. The properties of a Plackett-Burman design can be summarized as follows:

- By conducting n experiments (where n is a multiple of 4), $n - 1$ variables are screened – assuming first-order factor effects only (hence, assuming the lack of factor correlations);
- In most Plackett-Burman designs (including

those considered for our study) the factors are tested at $l = 2$ levels – indicated in a binary way as "+" and "−"; for numerical factors, these levels indicate "high" and "low" values, respectively;

- In the experiments, each factor is set $n/2$ times to "+" and $n/2$ times to "−";
- In order to enable statistical calculations, one or more so-called dummy variables are taken into account;
- Results are evaluated using a Student's t-test.

We adopted a $(n = 12, l = 2)$ Plackett-Burman design, in which the binary levels are as listed in Table 1.

The definition of the factors and their levels are listed in Table 2; roulette selection was used (see module **2** in the flowchart). The following factors require some further explanation (see [46, 44] for more details): x_3 (fitness offset) is a control parameter used in fitness translation – the first fitness scaling step usually applied to the raw fitness (see modules **0** and **1** in the flowchart); after fitness translation, the fitness of the worst-performing string equals the fitness offset; x_4 (fitness scaling factor) is another handle to affect fitness ratios; this factor is either the sensitivity in static linear fitness scaling mode or the segregation degree in sigmoid fitness scaling mode, whichever applies; both modes basically stabilize the competition between strings in roulette selection; x_6 (fraction elitism) is the fraction of strings which enjoy the privilege of guaranteed survival (reproduction); in GCA, these strings are additionally "immunized" (protected) against modification by recombination and mutation.

The experiments for the design were carried out using data set `dat15.x`. Due to the probabilistic nature of genetic algorithms, the response is distributed. Therefore, in order to get a rough impression about the reliability of the results, each experiment was carried out in duplo. A summary of the results is given in the next section; details of the calculations involved are beyond the purposes of this paper, as they can be derived from the theory in the cited literature on Plackett-Burman designs.

The configuration thus obtained was not only used in runs for data set `dat15.x`, but also in runs for data sets `dat25.x` and `dat50.x`, assuming that optimal configuration does not depend dramatically on the choice of the data set; this assumption seems reasonable in view of the widespread consensus among practitioners that genetic algorithms are configurationally robust [45, 46]. (The chosen data set is considered a legitimate part of the configuration, because it determines the parameterization of the objective function – module **0** in the flowchart.)

In Section 5.1 below, the configuration found is corroborated in three ways. One of these is the use of an additional, $(n = 8, l = 2)$ Plackett-Burman design involving x_2, x_4, x_5, x_7, x_8, x_9 (and a dummy factor) – the factors that were observed to have the most indeterminate impact in the first Plackett-Burman design.

At first glance, a legitimate point of objection against the pursued comparison between CLARA and GCA is that the running time of CLARA (used in standard configuration) is much smaller than the running time of GCA. In principle, one way to make the comparison more fair, would be to configure GCA for less string evaluations. However, this approach is not practically feasible since genetic algorithms are "heavy duty" optimizers; that is, at least in the order of 1000 to 10000 string evaluations must typically take place in many cases before genetic search acquires enough "momentum" to commence searching efficiently. For this reason, we decided to ignore the disparity between the running times in the first instance; this makes sense in view of the pilot nature of our study and of the fact that it can not be *a priori* ruled out that CLARA finds better solutions despite the shorter running time. In the second instance, if these investigations would reveal that GCA finds better solutions, then, in order to make the comparison more fair, we would decide to conduct additional runs with CLARA configured for running times closer to those of GCA.

5 Results and discussion

5.1 Configuration

The results of the experiments according to the $(n = 12, l = 2)$ Plackett-Burman design lead to the configuration listed in Table 3; the result $x_1 = 200$ has been omitted from this table.

The spread we observed in the response warned us that this configuration needs further corrobora-

$x_i \rightarrow$	x_1	x_2	x_3	x_a	x_4	x_5	x_6	x_b	x_7	x_8	x_9
experiment ↓ 1	+	+	−	+	+	+	−	−	−	+	−
2	−	+	+	−	+	+	+	−	−	−	+
3	+	−	+	+	−	+	+	+	−	−	−
4	−	+	−	+	+	−	+	+	+	−	−
5	−	−	+	−	+	+	−	+	+	+	−
6	−	−	−	+	−	+	+	−	+	+	+
7	+	−	−	−	+	−	+	+	−	+	+
8	+	+	−	−	−	+	−	+	+	−	+
9	+	+	+	−	−	−	+	−	+	+	−
10	−	+	+	+	−	−	−	+	−	+	+
11	+	−	+	+	+	−	−	−	+	−	+
12	−	−	−	−	−	−	−	−	−	−	−

Table 1: Binary levels for the $(n = 12, l = 2)$ Plackett-Burman design.

x_i	Control parameter	−	+
x_1	Number of generations	100	200
x_2	Population size	100	200
x_3	Fitness offset	0.0	0.02
x_a	dummy	irrelevant	irrelevant
x_4	Fitness scaling factor	1.5	2.5
x_5	Fitness scaling mode	static linear	sigmoid
x_6	Fraction elitism	0.0	0.01
x_b	dummy	irrelevant	irrelevant
x_7	Mix recombination probability	0.5	0.9
x_8	Point mutation probability	0.2	0.4
x_9	Mix mutation probability	0.0	0.125

Table 2: Levels considered in the $(n = 12, l = 2)$ Plackett-Burman design.

tion. This was done in the following three different ways. First, the abovementioned smaller($n = 8, l = 2$) Plackett-Burman design was conducted. This did not lead to new results. However, since recombination is normally the most important genetic exploration operator, we considered it worthwhile to examine four finer levels about 0.5, namely 0.3, 0.4, 0.6, and 0.7; from this fine-tuning it became apparent that level 0.4 is the best, and we switched to it. Second, we conducted some additional experiments for arbitrary combinations of binary levels other than those listed in Table 1. Again, no improvement was observed. Finally, we verified the levels for the recombination- and mutation operators by way of a so-called empirical complexity analysis [45, 44]. In such an analysis, a particular search heuristic of interest – the recombination- or mutation operator in our case – is allowed to perform an explorative walk in the search space. From the time-series of fitness values thereby obtained, one can derive (auto)correlation information indicating the ruggedness of the fitness landscape. This ruggedness is also called the apparent problem complexity; "apparent", as it is merely "perceived" by the search heuristic used. A lower apparent problem complexity is an indication of a better configuration of the search heuristic used. All combinations of binary levels for the recombination- and mutation- probabilities were tried, and the levels listed in Table 3 were again found to be the best.

Control parameter	Level
Population size	200
Fitness offset	0.02
Fitness scaling factor	2.5
Fitness scaling mode	static linear
Fraction elitism	0.01
Mix recombination probability	0.5
Point mutation probability	0.2
Mix mutation probability	0.125

Table 3: Levels obtained for the $(n = 12, l = 2)$ Plackett-Burman design.

5.2 Global sampling behavior

An impression of the global sampling behavior of a particular search method, as applied to a particular problem, can be obtained by measuring the frequency distribution of responses (or fitnesses) for the points sampled in the search space during a fixed-length run. Figure 3 depicts typical response frequency distributions obtained for random search (through a random walk) and GCA, respectively. The procedure is as follows. A range of feasible responses is subdivided into a number of small, equally sized intervals. The method of interest is then run for a fixed amount of string evaluations, and the occurences of responses falling within the respective intervals are counted in runtime; the count for each interval corresponds with a point in the response frequency distribution concerned. Useful information can be derived from response frequency distributions; for instance, from the response frequency distribution for random search in Figure 3 one may conclude that there are many non-optimal regions on the response landscape.

Response frequency distributions owe their appeal to being universal; that is, any pair of methods can be qualitatively compared through visual inspection of their respective response frequency distributions. A larger separation between the distributions indicates a larger difference in sampling efficiency, i.e. a larger difference in the ability to catch and exploit promising leads in the search space. Such a comparison is only meaningful, of course, if the number of string evaluations is chosen large enough. The 40000 evaluations used in order to obtain Figure 3 are considered enough, because for a sequence of increasing numbers of

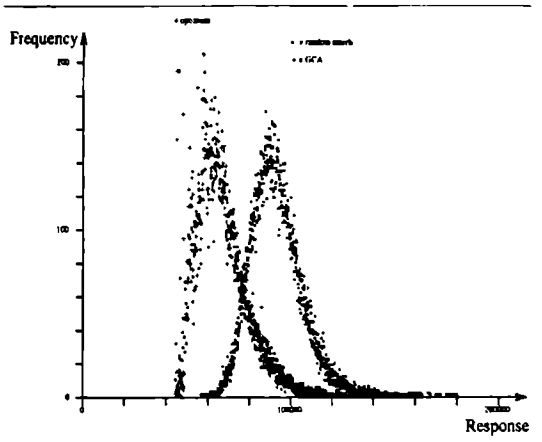

Figure 3: Response frequency distributions after 40000 string evaluations by random search and GCA, respectively, using `dat15.x`; the width chosen for the consecutive response intervals is 100 response units.

evaluations performed by either random search or GCA, we empirically established that the response frequency ratios converge at a number of evaluations well below 40000. (Convergence of this kind reflects a state of "dynamic equilibrium". The reason why a genetic algorithm, too, reaches such a dynamic equilibrium, is that it constantly escapes optima in searching for better optima – not "knowing" when the global optimum is reached.)

Comparisons between a genetic algorithm and random search are often described in the literature on genetic algorithms. The rationale for this is that a genetic algorithm, being probabilistic, can degrade to random search in the worst case; among plausible reasons why such may occur are a bad configuration or an inappropriate target problem. Hence, a plausible minimal condition for using a genetic algorithm is that it should perform better than random search. In this respect, response frequency distributions provide a useful prescreening tool to test whether mentioned minimal condition is fulfilled. It appears from Figure 3 that GCA passes such a test successfully.

5.3 Evolution of the population variance

In any genetic algorithm, sufficient population variance (diversity) must be maintained during the search for recombination to be productive [46].

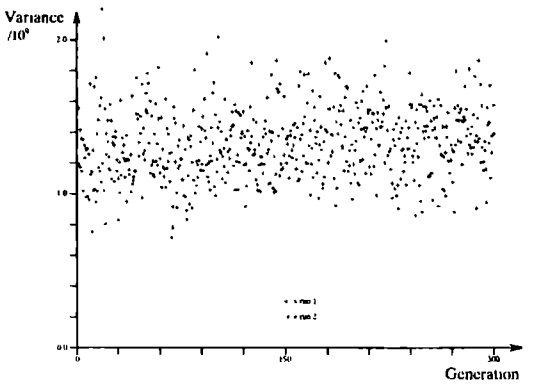

Figure 4: Evolution of the population variance, using `dat15.x` (duplo runs).

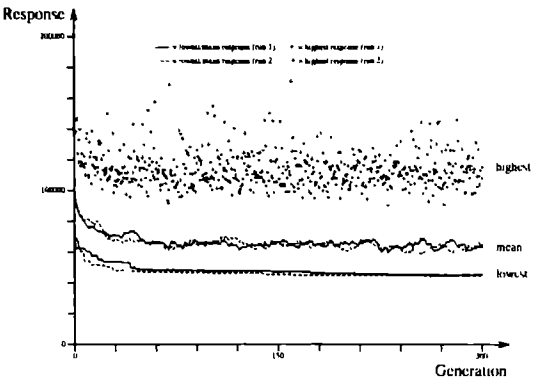

Figure 5: Evolution of the lowest-, highest- and mean response, using `dat15.x` (duplo runs).

Figure 4 illustrates the evolution of the population variance – measured as the variance in string responses for a population at a particular generation. (This approach is simple, yet provides a useful rough estimate since strings must be different if their responses are different; note that the opposite is not necessarily true: different strings may have the same response). Similar results were obtained for `dat25.x` and `dat50.x`. Added to the fact that the search converges (see next section), these results confirm that the configuration employed is acceptable. For instance, despite the strong selection pressure (exploitation) arising from elitism, a proper balance between exploration and exploitation is apparently maintained (since the average variance appears to remain constant).

5.4 Evolution of the response

It is informative to keep track of the lowest- (best-), highest- (worst-) and mean response in the population as it evolves (Figure 5). In this figure, it can be seen that the highest response evolves in a near-random way; this indicates that the genetic algorithm samples remote "corners" of the search space, which is desirable in order that local optima can be escaped. Nonetheless, as may be inferred from the way in which the mean response evolves and as is indeed to be generally expected, by far more search effort arises from the better strings in the population. The evolution of the lowest response exhibits the lowest noise level, which is also generally to be expected; moreover, note that the lowest response never rises, which is to be attributed to elitism.

Figure 6 shows in detail the evolution of the lowest response for the three data sets of interest. In these figures, the impact of a second genetic optimization step, using a modified configuration, is also shown; incidentally, such strategies wherein it is aimed to improve the result of a genetic algorithm by a second, differently configured genetic algorithm have been called sequential self-hybridization [46]. The configuration was modified as follows. Starting from the knowledge that, in general, recombination becomes less productive upon convergence and mutation then becomes a comparatively more important exploration operator accordingly, the mix recombination probability was changed from 0.4 to 0.0 (rendering the built-in mix mutation probability meaningless, of course); the point mutation probability was changed from 0.2 to 1.0; since this amounts to a stronger exploration, the level of exploitation was also enhanced (in order to maintain a proper balance between exploration and exploitation) by changing the fraction elitism from 0.01 to 0.02, and by switching from roulette selection to rank-based threshold selection [46, 44]; in this selection mode, strings with a fitness rank below some predefined threshold stand no chance of survival, i.e. are deterministically purged from the population – rendering the fitness scaling factor (x_4) and fitness scaling mode (x_5) meaningless.

Upon visually inspecting and comparing the three graphs shown in Figure 6, one may draw the following important conclusions. First, it is

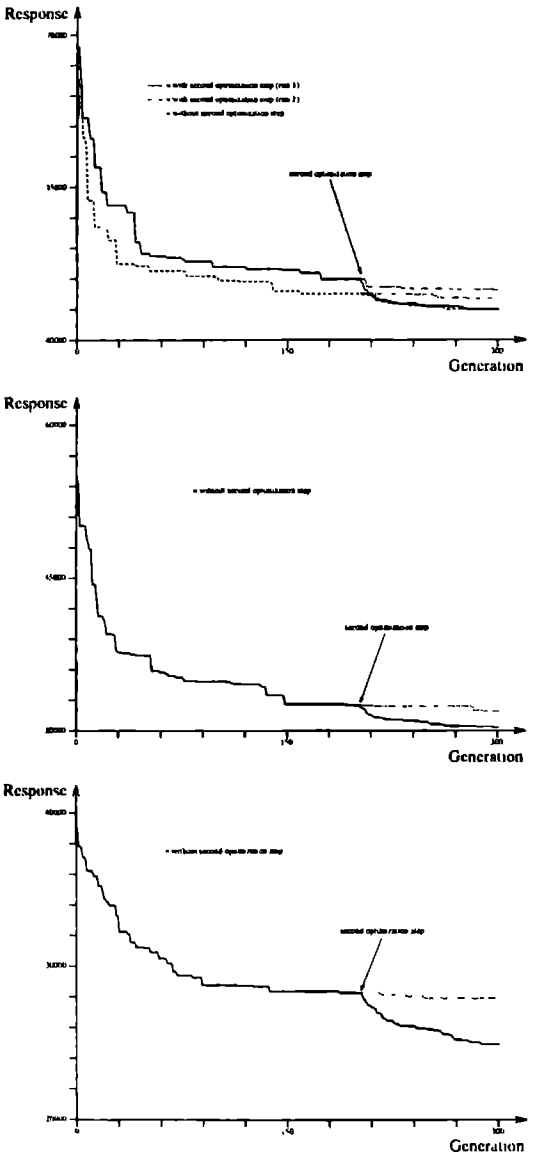

Figure 6: Evolution of the lowest response. Top: `dat15.x` (duplo runs). Middle: `dat25.x`. Bottom: `dat50.x`.

evident that a second optimization step can lead to a significant improvement; incidentally, this underscores the general intuition that if genetic configuration is to stay optimal during genetic search, then, ideally, it should evolve dynamically in response to the evolving population; nonetheless, only few applications of dynamically evolving genetic configurations have been reported up to now

D_k (sum of distances)				
data-set	CLARA		GCA	
`dat15.x`	59.208	67.9976	48.6959	48.769
`dat25.x`	45.0311	49.3854	36.5417	36.7439
`dat50.x`	40 2195	39.0047	30.8936	31.0254

Table 4: Sum of distances for end solutions (duplo runs).

(see e.g. [5, 37]), as it is apparently difficult to find generally successful strategies in practice. Second, convergence is reasonably approached within 300 generations, i.e. within the maximum allocated running time. Related to that, third, from the evolution of the lowest response it appears that the time constant of the convergence process is approximately 30 generations for all data sets, i.e. not strongly dependent on k. This suggests that GCA's time complexity in k is small, which is a valuable result since the end solutions are acceptable (as follows from the next section). In other words, the schema theorem (see Section 3.3.4 above) seems to apply in the cases studied, indicating that GCA recognizes implicit structure in data spaces quickly. However, it should be borne in mind that the used data sets are ideal (with hyperspherical, non-overlapping clusters).

In contrast, for CLARA the time constant of convergence – estimated from similar response evolution plots – was found to grow considerably faster with increasing k. (These results are not shown since the solutions found by CLARA are inferior to those found by GCA, as follows from the next section.)

5.5 Comparison of CLARA and GCA

In this section, we compare CLARA and GCA by the validation criteria outlined in Section 1.

5.5.1 Quality of the end solution

For the best solutions found in two runs with GCA (with the second optimization step) and two runs with CLARA, the values of D_k (decoded from D_k^*, in case of GCA, using Equation 5 after the runs) are listed in Table 4.

These results indicate that GCA finds significantly better clusterings, on average. Also, GCA seems more robust (reliable) as the spread in its

results is smaller, although this may also be a consequence of the larger running time.

It makes sense to extend the above comparison with a visual inspection of graphical plots of the clusters found, as it can not be generally ruled out that two different sets of medoids with different D_k result in the same clustering; this is especially likely the case when clusters are well separated. Such plots are shown in Figure 7 and Figures 8 for `dat15.x` and `dat50.x`, respectively.

From Figure 7 it follows that both CLARA and GCA generally manage to find all clusters in `dat15.x`. CLARA seems to have somewhat more difficulty with objects that lie between clusters, but since the computation time is much shorter, this is a price many practitioners would probably be willing to pay.

From Figure 8 it follows that CLARA does not succeed in finding all clusters in `dat50.x`, as some are joined together while others are split up; such performance is likely to be deemed unsatisfactory by many practitioners. GCA, on the other hand, still finds all clusters, although now it too seems to have some difficulty with objects that lie between clusters.

5.5.2 Computation time

From our experiments it follows that GCA requires considerably more running time in order to attain convergence. In order to make the comparison more fair, we examined whether CLARA would perform better when it is configured differently such that its computation time is larger (but still within the maximum allocated running time). This can be done in several ways. Recall that CLARA performs M major iterations, in each of which PAM performs a clustering of $K' = Ak + B < K$ objects – with 5, 2 and 40 as standard values for M, A and B, respectively. Hence, the present aim would be served by increasing M or K'. Both approaches were pursued, using the three artificial data sets available. We suffice with the following summary of the results.

Larger M values (leading to a linear increase in the computation time) generally did not improve the results significantly. Indeed, for M values that correspond with computation times approximately equal to the running time of GCA, the latter still performed considerably better for all data sets.

On the other hand, larger K' values (leading to a quadratic increase in the computation time) generally improved the results markedly, which is in agreement with results reported by Hopke and Kaufman [19]. For K' values that correspond with computation times approximately equal to the running time of GCA, CLARA performed virtually as good as GCA for `dat15.x` and almost as good for `dat25.x`, but by far worse for `dat50.x`. It is therefore concluded that for larger k values (i.e. for larger, more complex clustering problems), CLARA is increasingly prone to run into a sub-optimal solution from which it can not escape. We ascribe this shortcoming to the apparently greedy search heuristics used in both the minor cycle (PAM) and major cycle – in either cycle each step starting deterministically from the best that was found in the preceding step.

5.5.3 Memory usage

For both CLARA and GCA, memory usage is well within the memory specifications of the hardware used. Both programs save considerably on computer memory usage, as they store **D** in reduced form. GCA uses approximately 50 to 100 kB more computer memory, accounting for a somewhat larger program code and a population.

5.5.4 Ease of development and implementation

If developed from scratch, the programming and debugging of CLARA and GCA consumes approximately equal effort. In our specific case, GCA was obtained with minimal effort since we borrowed a genetic algorithm prototype from a genetic software library (abovementioned GATES) that was available. On the other hand, however, the optimization of GCA is considerably more laborious due to the larger amount of (correlated) control parameters.

5.5.5 Ease of use

Once implemented and optimized, CLARA and GCA feature comparable ease of use. If wider distribution among routine users is intended, a more advanced user interface is desirable in both programs.

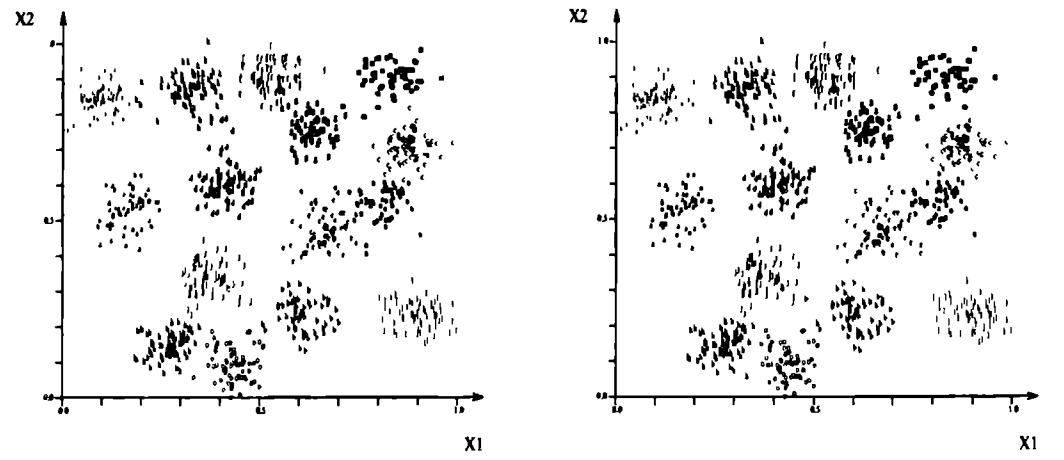

Figure 7: Clustering found for data set `dat15.x`. Left: CLARA. Right: GCA. (*Cf.* Figure 2-top.)

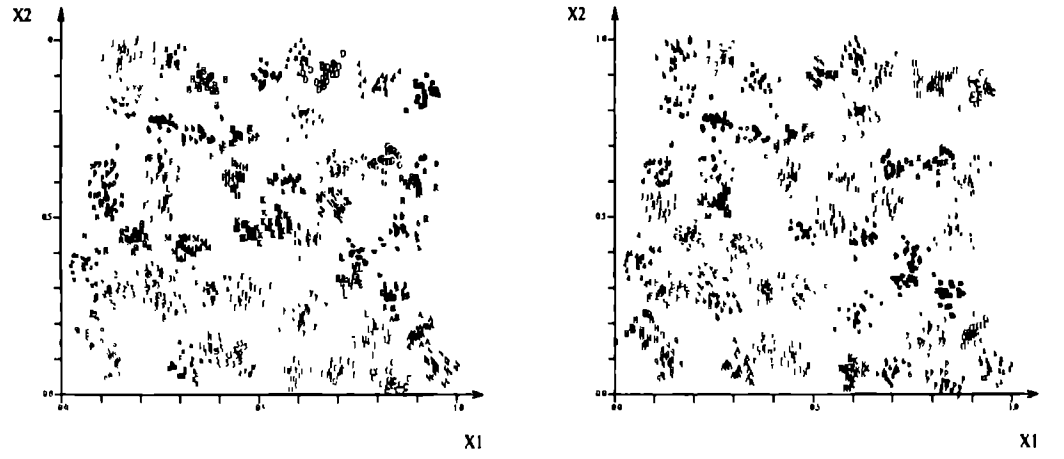

Figure 8: Clustering found for data set `dat50.x`. Left: CLARA. Right: GCA. (*Cf.* Figure 2-bottom.)

6 Conclusions and outlook

A pilot study has been presented that treats the background, feasibility and competitiveness of GCA – a genetic algorithm for k-medoid partitional clustering of large data sets. In general, GCA turns out to be practically feasibe for problems of this kind, as it finds acceptable solutions in reasonable time.

The competitiveness of GCA was rated with respect to CLARA, an earlier approach to k-medoid clustering of large datasets. This comparison has been carried out using artificial data sets with a known, ideal structure – hence with known k – for verification purposes.

For small values of k, both CLARA and GCA find acceptable solutions. Those found by GCA are only slightly better at the expense of a considerably larger amount of computation time. However, with increasing values of k, the picture changes drastically: the computation time of CLARA increases quadratically; in addition, the quality of the solutions found drops rapidly below acceptable. The computation time of GCA, on the other hand, increases much slower and acceptable solutions are still found. These results indicate that GCA is a better choice when many clusters are present in the target data set. Whether this

conclusion still applies when non-ideal, real-world data sets are used, is an important subject for future investigation.

A fruitful avenue for further research seems to lie in the sequential hybridization [46] of GCA with another method to enhance performance. We demonstrated that self-hybridization – in our case, a second optimization step to GCA by GCA itself – comprises a viable approach. In a similar way, GCA can be combined with CLARA to obtain a hybrid searching system with better performance. For instance, GCA can be post-hybridized with CLARA by passing the solution found by GCA to CLARA for further improvement; this demands that CLARA is customized such that in the first major iteration the sample of objects passed to PAM contains the set of medoids found by GCA. Also, GCA can be pre-hybridized with CLARA by seeding the (otherwise normally random) initial population of GCA with a few solutions found by CLARA. Preliminary results that we have obtained with such post- and pre-hybridization strategies are encouraging and underscore the viability of hybrid searching systems in general.

In principle, GCA lends itself excellently for parallelization [60], as each string in the population can be evaluated independently from the other strings. When parallelized, the computation is distributed across a host processor and several slave processors; most of the code for GCA is loaded on the host processor (except for the objective function), while each slave processor accommodates a small sub-population – possibly as small as one string - and the code for the objective function to evaluate the strings therein. In contrast, CLARA is more difficult to parallelize because the successive major iterations are not mutually independent [26, 18]; moreover, memory demands are higher, as PAM needs to be loaded on each of the slave processors. Importantly, the parallelization of GCA can be accomplished without specialized parallel hardware such as transputer systems that make memory access and host-slave communication fast, because the overall speed of execution is mostly determined by string evaluations. An example of a parallel system of processors that would suffice, is a local area network of workstations; the software required for the communication is comparatively simple and cheap.

References

[1] Aarts, E.H.L. and Korst, J.H.M. *Simulated Annealing and Boltzmann Machines. A Stochastic Approach to Combinatorial Optimization and Neural Computing.* Wiley, Chichester, 1989.

[2] Belew, R.K. and Booker, L.B., editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann

[3] Bhuyan, J.N., Raghavan, V.V., and Elayavalli, V.K. Genetic algorithms for clustering with an ordered representation. In Belew, R.K and Booker, L.B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 408–415, San Mateo, CA, 1991. Morgan Kaufmann.

[4] Davidor, Y. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization.* World Scientific, Singapore, 1991.

[5] Davis, L. Adapting operator probabilities in genetic algorithms. In Schaffer, J.D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69, San Mateo, CA, 1989. Morgan Kaufmann.

[6] Davis, L., editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, NY, 1991.

[7] Dubes, R. and Jain, A.K. Clustering methodologies in exploratory data analysis. In Yovits, M.C., editor, *Advances in Computers*, pages 113–215, New York, NY, 1980. Academic Press. Volume 19.

[8] Falkenauer, E. A genetic algorithm for conceptual clustering. Technical Report Report FMS39, CRIF Industrial Automation, Brussels, December 1991.

[9] Falkenauer, E. A genetic algorithm for grouping. In Gutiérrez, R. and Valderrama, M J., editors, *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis*, pages 198–206, Singapore, 1991. World Scientific.

[10] Glover, F. Tabu search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[11] Glover, F. Tabu search – Part II. *ORSA Journal on Computing*, 2·4–32, 1990.

[12] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[13] Grefenstette, J.J., editor. *Proceedings of the First International Conference on Genetic Algorithms*, Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[14] Grefenstette, J J , editor *Proceedings of the Second International Conference on Genetic Algorithms*, Hillsdale, NJ, 1987 Lawrence Erlbaum Associates

[15] Hansen, P , Jaumard, B , and Da Silva, S Average linkage divisive hierarchical clustering *Journal of Classification*, 1993 To appear

[16] Holland, J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[17] Holland, J H Genetic algorithms *Scientific American*, 267(1) 44–50, July 1992

[18] Hopke, P K The application of supercomputers to chemometrics In Karjalainen, E J , editor, *Scientific Computing and Automation (Europe) 1990*, pages 9–19, Amsterdam, 1990 Elsevier

[19] Hopke, P K and Kaufman, L The use of sampling to cluster large data sets *Chemometrics and Intelligent Laboratory Systems*, 8 195–204, 1990

20] Jain, A K and Dubes, R C *Algorithms for Clustering Data* Prentice-Hall, Englewood Cliffs, NJ, 1988

21] Jones, D R and Beltramo, M A Solving partitioning problems with genetic algorithms In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 442–449, San Mateo, CA, 1991 Morgan Kaufmann

[22] Jones, K Optimization of experimental data *International Laboratory*, pages 32–45, November 1986

[23] Jurs, P C *Computer Software Applications in Chemistry* Wiley, New York, NY, 1986

[24] Jurs, P C and Isenhour, T L *Chemical Applications of Pattern Recognition* Wiley, New York, NY, 1975

[25] Kamgar-Parsi, B , Gualtieri, J A , Devaney, J E , and Kamgar-Parsi, B Clustering with neural networks *Biological Cybernetics*, 63 201–208, 1990

[26] Kaufman L , Hopke, P K , and Rousseeuw, P J Using a parallel computer system for statistical resampling methods *Computational Statistics Quarterly*, 2 129–141, 1988

[27] Kaufman, L and Rousseeuw, P Clustering large data sets In Gelsema, E and Kanal, L , editors, *Pattern Recognition in Practice II*, pages 425–437, Amsterdam, 1986 Elsevier (with discussion)

[28] Kaufman, L and Rousseeuw, P J *Finding Groups in Data An Introduction to Cluster Analysis* Wiley Chichester, 1990

[29] Kirkpatrick, S , Gelatt, C D Jr , and Vecchi, M P Optimization by simulated annealing *Science*, 220 671–680, 1983

[30] Klein, R W and Dubes, R Experiments in projection and clustering by simulated annealing *Pattern Recognition*, 22 213 220, 1989

[31] Koontz, W L G , Narendra, P M , and Fukunaga, K A branch and bound clustering algorithm *IEEE Transactions on Computers*, C-23 908–914, 1975

[32] Koza, J R *Genetic Programming On the Programming of Computers by Means of Natural Selection* MIT Press, Cambridge, MA, 1992

[33] Laarhoven van, P J M and Aarts, E H L *Simulated Annealing Theory and Applications* Mathematics and its Applications (East European Series) D Reidel, Dordrecht, 1987

[34] Land, A H and Doig, A G An automatic method of solving discrete programming problems *Econometrica*, 28 497–520, 1960

[35] Laszewski von, G Intelligent structural operators for the k-way graph partitioning problem In Belew, R K and Booker, L B , editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45 52, San Mateo, CA, 1991 Morgan Kaufmann

[36] Leardi, R , Boggia, R , and Terrile, M Genetic algorithms as a strategy for feature selection *Journal of Chemometrics*, 6 267–281, 1992

[37] Li, T -H , Lucasius, C B , and Kateman, G Optimization of calibration data with the dynamic genetic algorithm *Analytica Chimica Acta*, 268(1) 123–134, 1992

[38] This paper

[39] Lucasius, C B , Beckers, M L M , and Kateman, G Genetic algorithms in wavelength selection A comparative study *Analytica Chimica Acta*, 1993 Submitted

[40] Lucasius, C B , Kampen van, A H C , Buydens, L M C , and Kateman, G On the robustness of genetic algorithms in noisy optimization An application in non-linear compaction of variates *Chemometrics and Intelligent Laboratory Systems*, 1993 In preparation

[41] Lucasius, C B and Kateman, G Genetic algorithms for large-scale optimization in chemometrics An application *Trends in Analytical Chemistry*, 10 254–261, 1991

[42] Lucasius, C B and Kateman, G Towards solving subset selection problems with the aid of the genetic algorithm In Manner R and Manderick,

B , editors, *Proceedings of the Second Workshop on Parallel Problem Solving from Nature*, pages 239–247, Amsterdam, 1992 Elsevier

[43] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 1 General perspective *Computers & Chemistry*, 1993 Accepted

[44] Lucasius, C B and Kateman, G GATES towards evolutionary large-scale optimization A software-oriented approach to genetic algorithms Part 2 Toolbox description *Computers & Chemistry*, 1993 Accepted

[45] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 1 Concepts, properties and context *Chemometrics and Intelligent Laboratory Systems*, 19 1–33, 1993

[46] Lucasius, C B and Kateman, G Understanding and using genetic algorithms Part 2 Representation, configuration and hybridization *Chemometrics and Intelligent Laboratory Systems*, 1993 In press

[47] Maffioli, F The complexity of combinatorial optimization problems and the challenge of heuristics In Christofides, N , Mingozzi, A , Toth, P , and Sandi, C , editors, *Combinatorial Optimization*, page Chapter 5, New York, NY, 1979 Wiley

[48] Massart, D L and Kaufman, L *The Interpretation of Analytical Chemical Data by the Use of Cluster Analysis*, volume 65 of *Chemical Analysis* Wiley, New York, NY, 1983

[49] Massart, D L , Plastria, F , and Kaufman, L Non-hierarchical clustering with MASLOC *Pattern Recognition*, 16 507–516, 1983

[50] Massart, D L , Vandeginste, B G M , Deming, S N , Michotte, Y , and Kaufman, L *Chemometrics A textbook* Elsevier, Amsterdam, 1988

[51] Melssen, W J , Smits, J R M , Rolf, G H , and Kateman, G Two-dimensional mapping of IR spectra using a parallel implemented self-organising feature map *Chemometrics and Intelligent Laboratory Systems*, 1993 In press

[52] Michalewicz, Z *Genetic Algorithms + Data Structures = Evolution Programs* Artificial Intelligence Series Springer-Verlag, Berlin, 1992

[53] Milligan, G W An algorithm for generating artificial test clusters *Psychometrika*, 50(1) 123–127, 1985

[54] Reeves, C R , editor *Modern Heuristic Techniques for Combinatorial Problems* Advanced Topics in Computer Science Blackwell Scientific Publications, Oxford, 1993

[55] Rousseeuw, P J Silhouettes A graphical aid to the interpretation and validation of cluster analysis *Journal of Computational and Applied Mathematics*, 20 53–65, 1987

[56] Schaffer, J D , editor *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989 Morgan Kaufmann

[57] Sharaf, M A , Illman, D L , and Kowalski, B R *Chemometrics*, volume 82 of *Chemical Analysis* Wiley, New York, NY, 1986

[58] Siedlecki, W and Sklansky, J A note on genetic algorithms for large-scale feature selection *Pattern Recognition Letters*, 10 335–347, 1989

[59] Smits, J R M , Melssen, W J , Buydens, L M C , and Kateman, G Using artificial neural networks for solving chemical problems A tutorial *Chemometrics and Intelligent Laboratory Systems*, 1993 Accepted

[60] Stender, J , editor *Parallel Genetic Algorithms Theory and Applications* Frontiers in Artificial Intelligence and Applications IOS Press, Amsterdam, 1993

[61] Varmuza, K *Pattern Recognition in Chemistry*, volume 21 of *Lectures Notes in Chemistry* Springer Verlag, New York, NY, 1980

[62] Wehrens, R , Lucasius, C B , Buydens, L , and Kateman, G HIPS, a hybrid self-adapting expert system for NMR spectrum interpretation using genetic algorithms *Analytica Chimica Acta*, 277 313–324, 1993

[63] Wehrens, R , Lucasius, C B , Buydens, L , and Kateman, G Sequential assignment of 2D NMR spectra of proteins using genetic algorithms *Journal of Chemical Information and Computer Sciences*, 33(2) 245–251, 1993

[64] Zupan, J *Algorithms for chemists* Wiley, New York, NY, 1989

CHAPTER 13

Genetic Algorithms in Wavelength Selection: A Comparative Study

Contents

Genetic Algorithms in Wavelength Selection: A Comparative Study

Abstract

This paper presents a comparative study involving a genetic algorithm, simulated annealing, and stepwise elimination, as methods for wavelength selection in multi-component analysis. The wavelength selection criteria used are the selectivity and accuracy after Lorber, and the minimal mean squared error after Sasaki. The genetic algorithm generally performed best. Stepwise elimination performed surprisingly good despite its local search heuristic. Simulated annealing performed worst, which is remarkable in view of the fact that this method is widely praised in the literature for properties similar to those of genetic algorithms, e.g. a probabilistic, non-local search heuristic.

1 Introduction

Multi-component analysis (MCA) is the simultaneous determination of analyte concentrations in a mixture of components (analytes) from their spectra or absorption curves by means of a least squares fit, assuming that Beer-Lambert's law holds. It may greatly speed up the overall analysis time and therefore enjoys much attention. However, poorly defined effects – e.g. overlap of spectral bands, component interactions, deviations from Beer-Lambert's law at high absorbances, etc. – may lead to a bad *accuracy* and *precision*.

A frequently applied strategy to improve the accuracy, is wavelength selection based on criteria such as selectivity and/or sensitivity [38], discussed below; the quality of the data is improved, hence more accurate concentration estimates are obtained.

In order to improve the precision, other criteria should be should be used, e.g. the minimum mean squared error [39], discussed below; the quality of the model is improved, hence more precise concentration estimates are obtained.

The precision generally becomes better as more wavelengths are selected; the best precision is attained when all wavelengths in the spectrum are selected. However, this is accompanied by a loss in accuracy. Thus, since accuracy and precision are two competing objectives, the best compromise between both is often sought through simultaneous optimization.

The next subsections provide an overview of the methods and criteria used in this comparative study, and explain the complexity of the wavelength selection problem, in general.

1.1 Methods for wavelength selection

Various methods for wavelength selection have been proposed in the literature [17, 18, 14, 15, 2, 39, 33]. Recently, we demonstrated the viability of genetic algorithms (GAs) in tackling wavelength selection problems [27]. The present paper may be regarded as a continuation of this work. In particular, it addresses the question that should logically be asked next: where do GAs stand relative to other search techniques applied in wavelength selection? To obtain a first indication of the answer to this question, a comparison is made with two other search techniques that have been applied before in wavelength selection:

- Stepwise elimination (SE) [14];
- Simulated annealing (SA) [17, 16, 18].

Both SE and SA enjoy considerable interest, and are simple to implement.

SE, which probably enjoys the longest tradition of application, is based on a deterministic, local ("greedy") search heuristic. Local search is generally deemed as sensitive to getting trapped in local optima, but may nevertheless turn out to be adequate, depending on the nature of the problem.

SA, which has more recently gained popularity, is based on a probabilistic, non-local search heuristic and is therefore generally regarded as robust, i.e. insensitive to getting trapped in local optima. Since GAs, too, are based on a probabilistic, non-local search heuristic, the comparison between the GA and SA may be considered particularly important.

In the remainder, the following notations are regularly used:

n : number of components contained in the mixture subjected to MCA;
m : number of wavelengths selected from the spectrum, to conduct MCA;
M : number of wavelengths contained in the spectrum.

In this way, the wavelength selection problem can be defined as the problem to select, according to some predefined criterion, the optimal subset of m wavelengths among the M wavelengths contained in the spectrum, where $n \leq m \leq M$.

1.2 Complexity of wavelength selection

This section briefly inspects the complexity of wavelength selection problems, and how different search strategies deal with that.

Qualitatively, the complexity of wavelength selection problems may be phrased as follows: they can not be solved in an analytical (i.e. deductive) way, hence a search-based (i.e. inductive) strategy, or optimization, is needed. Abovementioned SE, SA, and GA are, of course, just a few examples of search strategies that can be used.

A more quantitative definition of the complexity of wavelength selection problems considers the formal size of the search space, i.e. the collection of all possible subsets of m wavelengths that can be drawn from the source set of M wavelengths. Thereby, it is important to realize that wavelength selection problems belong to the more general class of subset selection problems, and therefore may have certain properties that some subset selection problems are known to have. One such property is $\mathcal{NP}$-completeness [7, 32]. An $\mathcal{NP}$-complete problem is a problem that features the highest level of complexity in that no other problem exists for which it is more difficult to find an algorithm that solves the problem in $O(M^{\alpha})$, or polynomial, time. (M is the problem dimensionality, or, in our case, the number of wavelengths contained in the spectrum; α is a non-negative constant.) That is, any proposed algorithm is likely to solve the problem in $O(\alpha^{M})$, or exponential, time. Strickly, "solving the problem" means that the globally optimal solution, or global solution for short, is found. In many cases, however, it is practically difficult or even impossible to verify whether a particular solution represents the global solution, but it may still be considered acceptable. As a rule, adequate local solutions are found in a considerably shorter time. Also, a particular algorithm that finds the global solution in exponential time, may find adequate local solutions in polynomial time.

A simple example of an exponential time algorithm in wavelength selection, is enumerative search (i.e. systematic, or exhaustive, search). All possible subsets are scanned – i.e. approximately 2^{M} subsets when $n \leq m \ll M$, as usually is the case. In this way, enumerative search guarantees that the globally optimal subset will be found. However, its major drawback is that even for moderately sized data sets (typically $100 \leq M \leq 200$), the time required to obtain the result lies far beyond what is practically feasible.

In order to achieve a running time within reasonable practical time limits, other methods scan a much smaller portion of the search space, but thereby always risk missing the global solution. For instance, SE (which, like other incremental stepping heuristics, e.g. [26], can easily be shown to have time complexity $O(M^2)$) samples an extremely tiny portion of the search space due to its local nature; this makes SE generally sensitive to local optima, as pointed out above. SA and GAs, on the other hand, are both generally praised for the fact that they search in a non-local way, yet may find adequate solutions in polynomial time [19, 11, 9].

1.3 Overview of criteria for subset evaluation

Search may be viewed as a sequence of intendedly improving guesses made at the global solution. In the wavelength selection problem these guesses are candidate subsets of wavelengths proposed by the search heuristic concerned. The candidate subsets are evaluated, i.e. assigned a score, by a predefined objective function, or optimality criterion, or *evaluation criterion*. The scores delivered by the evaluation criterion are used in order to give direction to the search.

(We should comment that the term "evaluation criterion" is not customary in the literature on wavelength selection; instead, one speaks of a "selection criterion". However, the term "selection criterion" also exists in GA jargon; there it has an entirely different meaning, as becomes clear in Section 3.2.1 below. Thus, the term "evaluation criterion" is used to avoid confusing ambiguous

use of the term "selection criterion" in this paper.)

The following evaluation criteria are applied in this study:

SEL (Section 2.2.1);
ACC (Section 2.2.3);
MMSE (Section 2.3).

1.4 Overview of criteria for method validation

Search methods are mutually compared on the basis of a *method validation criterion*. Various method validation criteria are conceivable, e.g.:

1. the quality of the solution found by the method;
2. the amount of computation time required by the method;
3. the amount of computer memory required by the method;
4. the ease of developing and implementing the method;
5. the ease of using the method.

For the purposes of this pilot study, mainly the first among these general method validation criteria is considered. It can be defined more specifically in different ways; the quality criteria used in this study are:

HIT (Section 3.3.1);
PE (Section 3.3.2);
RSD (Section 3.3.2).

As becomes apparent below, these criteria have in common that they verify the extent to which the solution found by the search method concerned, satisfies available foreknowledge about the problem domain.

2 Theory

This section discusses the following evaluation criteria for wavelength selection in MCA: SEL (selectivity) and ACC (accuracy) after Lorber [25] and MMSE (minimum mean squared error) after Sasaki *et al.* [39].

SEL is a measure of the degree of non-overlap between the different component spectra. The subset of wavelengths that features the lowest overall overlap (hence the highest overall selectivity) should yield the most accurate component concentration estimation.

MMSE is a measure of the variance in the concentration vector (originating from the expectation of the noise). In general, larger subsets of wavelengths yield a smaller MMSE, i.e. a better precision in the concentration estimates; simultaneously, though, the accuracy in those estimates becomes worse, as pointed out above.

ACC is actually not a measure of accuracy (notwithstanding this name). It is in fact composed of two precisions, one for the calibration and the other for the total measurement. Like MMSE, it promotes the selection of large subsets of wavelengths to achieve a better precision in the concentration estimates.

Summarizing, SEL is a measure of accuracy, while MMSE and ACC are measures of precision.

From here on, the following notations are used. Matrices are denoted by uppercase characters. Vectors are denoted by lowercase characters. The superscripts T and t denote transposed matrices and vectors, respectively. $\|\mathbf{x}\|$ is the Euclidian norm of arbitrary vector $\mathbf{x}$; it is calculated as the square root of the sum of the squared elements. $\mathbf{I}$ is the identity matrix. The subscript i denotes a selected wavelength ($i = 1, \cdots, m$). The subscript j denotes a component ($j = 1, \cdots, n$).

2.1 Multi-component analysis

For the determination of component concentrations in a mixture, the linear additive model after Beer-Lambert is frequently applied:

$$\begin{aligned} d_1 &= a_{11} \cdot c_1 + a_{12} \cdot c_2 + \cdots + a_{1n} \cdot c_n \\ d_2 &= a_{21} \cdot c_1 + a_{22} \cdot c_2 + \cdots + a_{2n} \cdot c_n \\ &\vdots \\ d_m &= a_{m1} \cdot c_1 + a_{m2} \cdot c_2 + \cdots + a_{mn} \cdot c_n \end{aligned}$$

where $\mathbf{d}_i$ is the measured absorbance at wavelength i, and a_{ij} is the known absorption coefficient at wavelength i for component j. Using matrix notation, the linear system of equations can be abbreviated as:

$$\mathbf{d} = \mathbf{S} \cdot \mathbf{c}$$

or:

$$\mathbf{d} = \mathbf{A} \cdot \mathbf{C_0}^{-1} \cdot \mathbf{c} \quad (1)$$

where:

$\mathbf{d}$		m-dimensional response vector (absorbances);
$\mathbf{c}$	:	n-dimensional concentration vector;
$\mathbf{S}$	:	$m \times n$ sensitivity matrix;
$\mathbf{A}$	:	$m \times n$ scaled sensitivity matrix (molar absorption coefficients) or calibration matrix;
$\mathbf{C_0}^{-1}$	:	$n \times n$ diagonal matrix of inverse concentrations ($c_{0,i}$) that correspond to the absorption coefficients in $\mathbf{A}$.

For the moment, assume $m = M$ (all wavelengths selected). Finding the solution of Equation 1 is particularly easy when every component has a strong absorbance peak at a particular wavelength, without interference of the other components at that wavelength. This is for instance the case in a two-component system ($n = 2$) where component j_a absorbs strongly at wavelength i_a but hardly at wavelength i_b, whereas component j_b absorbs strongly at wavelength i_b but hardly at wavelength i_a. At these two wavelengths there is (almost) no overlap of the absorption curves, hence the individual selectivities, and thus the overall selectivity, are high. Equation 1 can then be solved for the appropriate two wavelengths (i_a and i_b, $m = 2$) selected among the M wavelengths.

Similarly, in a highly selective n-component system, Equation 1 can be solved by selecting the appropriate $m = n$ wavelengths among the M wavelengths. However, many multi-component systems are poorly selective in practice (e.g. in UV spectrometry), as different components often absorb strongly at the same wavelengths. Equation 1 can then only be solved by using $m > n$ wavelengths among the M wavelengths, provided that the individual absorption curves are not linearly dependent.

Several factors can influence the mutual linear dependency of the individual absorption curves, e.g. *pH*, ionic strength, the concentration of salts or other interferents. It is generally not evident how these factors should be manipulated to reduce the linear dependency. On the other hand, wavelength selection based on criteria such as selectivity or sensitivity, can be used to select a subset of wavelengths in which the linear dependency between individual components is reduced as much as possible. By using this subset, the overlap between the individual absorption curves is reduced, hence the selectivity is maximal; or, phrased mathematically, the molar absorption coefficient vectors of the individual components approach orthogonality as much as possible.

The formal solution of Equation 1 is given by the least-squares fit (estimate):

$$\hat{\mathbf{c}} = (\mathbf{S}^T \cdot \mathbf{S})^{-1} \cdot \mathbf{S}^T \cdot \mathbf{d}$$

when $m = n$ or when $m > n$ and the rank of $\mathbf{S}$ (or $\mathbf{A}$) is n. However, this approach fails when the rank of $\mathbf{S}$ (or $\mathbf{A}$) is k, and $k < n$; the problem must then be solved using the $n \times m$ generalized inverse $\mathbf{S}^+$ (or $\mathbf{A}^+$) [36]:

$$\hat{\mathbf{c}} = \mathbf{S}^+ \cdot \mathbf{d}$$

or:

$$\hat{\mathbf{c}} = \mathbf{A}^+ \cdot \mathbf{C_0} \cdot \mathbf{d} \quad (2)$$

The generalized inverse is calculated by means of a singular value decomposition (SVD) [6, 8].

2.2 Figures of merit after Lorber

The figures of merit introduced by Lorber [25] are a series of quality features for an analytical method that include the error propagation, the signal/noise ratio, the limit of detection, the precision, the accuracy, the sensitivity, and the selectivity. These figures are all calculated using the *net analyte signal*, which is defined as that part of the spectrum of a component in a mixture that is orthogonal to the spectra of the other components in that mixture. (Only the orthogonal part is relevant, because the part of the spectrum of a particular component that is not orthogonal to the other spectra, is a linear combination of the spectra of the other components.)

The net analyte signal, $\mathbf{a}_j^*$, is defined as a multiplication of $\mathbf{a}_j$ with its projection matrix:

$$\mathbf{a}_j^* = (\mathbf{I} - \mathbf{A}_j \cdot \mathbf{A}_j^+) \cdot \mathbf{a}_j$$

where: $\mathbf{a}_j$ is the j^{th} column vector in $\mathbf{A}$; $\mathbf{A}_j$ is the "$\mathbf{A}$-without-$\mathbf{a}_j$" matrix, hence $m \times (n-1)$; $\mathbf{A}_j^+$ is the generalized inverse of $\mathbf{A}_j$.

Likewise:

$$\mathbf{d}_j^* = (\mathbf{I} - \mathbf{A}_j \cdot \mathbf{A}_j^+) \cdot \mathbf{d}_j$$

2.2.1 Selectivity and sensitivity

When the j^{th} row of $\mathbf{A}^+$ is denoted by $\mathbf{y}_j$, then for each component separately Equation 2 becomes $c_j = c_{0,j} \cdot \mathbf{y}_j^t \cdot \mathbf{d}$. The net analyte signal is now calculated as [25]:

$$\mathbf{a}_j^* = \frac{\mathbf{y}_j}{\|\mathbf{y}_j\|^2}$$

All figures of merit defined by Lorber are expressed in terms of this formulation of the net analyte signal or its vector norm. For instance, Lorber defines selectivity for the j^{th} component as:

$$\text{SEL}_j = \frac{\|\mathbf{a}_j^*\|}{\|\mathbf{a}_j\|} \tag{3}$$

The sensitivity for each individual components is now given by:

$$\text{SEN}_j = \frac{\|\mathbf{a}_j^*\|}{c_{0,j}} \tag{4}$$

The overall selectivity, i.e. the selectivity for the mixture, is defined as:

$$\text{SEL} = \frac{n}{\sum_{j=1}^{n} \text{SEL}_j^{-1}} \tag{5}$$

(The perhaps more straightforward definition $\text{SEL} = \sum_{j=1}^{n} \text{SEL}_j/n$ is not adequate, because the overall selectivity is dominated by the component with the worst selectivity.)

2.2.2 Precision

For $m > n$, the error in $\mathbf{d}$ (the response data) may be calculated as:

$$\epsilon_d = \left[\frac{\mathbf{d}^t \cdot (\mathbf{I} - \mathbf{A} \cdot \mathbf{A}^+) \cdot \mathbf{d}}{m-n}\right]^{\frac{1}{2}}$$

or estimated by [2]:

$$\epsilon_d = \left[\frac{(\mathbf{d} - \hat{\mathbf{d}})^2}{m-n}\right]^{\frac{1}{2}}$$

where $\hat{\mathbf{d}}$ is the estimated $\mathbf{d}$. For $m = n$, ϵ_d has to be determined in a different way, e.g. Malinowski's error function [33]. However, as the noise in the data may be expected to exhibit normal behavior (zero mean, constant variance), ϵ_d may be assumed constant. Zscheile [41] found $\epsilon_d = 2.0\%$ for a data that we also use in this study (see Section 3.1.1 below). Lorber found $\epsilon_d = 1.3\%$ for the same data set, but he continued to use 2.0% nevertheless [25].

The relative precision in $\mathbf{d}$ for the sample, is the ratio of ϵ_d to the total measurement $\|\mathbf{d}\|$.

2.2.3 Accuracy

The accuracy in an analytical quantity is influenced by several factors that cause deviation from its true value, namely:

1. the precision in the measured response of the unknown sample;
2. the precision in measuring the calibration data;
3. inadequacy of the assumed linear model to describe the true model.

To find an expression for the accuracy, Lorber [25] assumed that only the first two factors contribute to the accuracy. The accuracy can then be defined as the sum of the two contributing precisions. Hence, it is expected that upon using this evaluation criterion, the precision in the concentration estimation will improve; indeed, the term "accuracy" is not used in its proper sence, as pointed out above. Moreover, since factor 3 is not incorporated in the definition, it may be expected that systematic errors cannot be detected.

Assuming $\epsilon_d = \epsilon_a$ (error of calibration), it can be deduced [13] that Lorber's accuracy is given by:

$$\text{ACC}_j = \left(\frac{1}{\text{SEL}_j \cdot \|\mathbf{a}_j\|} + \frac{1}{\text{SEN}_j \cdot c_j}\right) \cdot \epsilon_d$$

where SEL_j and SEN_j are as defined in Equations 3 and 4, respectively.

The overall accuracy, i.e. the accuracy for the mixture, is defined as [15]:

$$\text{ACC} = \frac{\sum_{j=1}^{n} \text{ACC}_j}{n} \tag{6}$$

2.3 Minimum mean squared error after Sasaki

The minimum mean squared error criterion was used by Sasaki *et al.* [39]. It calculates the difference between the theoretical component concentrations and their estimates, hence the variance in the concentration vector, that originates from the expectation value of the noise.

Starting from Beer-Lambert's linear additive model $\mathbf{d} = \mathbf{A} \cdot \mathbf{c} + \mathbf{e}$, for which the solution $\hat{\mathbf{c}} = \mathbf{A}^{-1} \cdot \mathbf{d}$ applies, provided $\mathbf{A}$ is square and not singular, the minimum mean squared error is defined as:

$$\mathsf{MMSE} = E\left\{\sum_{j=1}^{n} \mid c_j - \hat{c}_j \mid^2\right\} \rightarrow min$$

where E is the expectation operator, and $\hat{c}_j$ is the estimated c_j. This equation can be rewritten as:

$$\mathsf{MMSE} = Tr\left\{\mathbf{A}^{-1} \cdot \mathbf{W}^{-1} \cdot (\mathbf{A}^{-1})^T\right\}$$

where Tr is the trace operator, and $\mathbf{W} = E\{\mathbf{e} \cdot \mathbf{e}^t\}$, the $n \times n$ autocorrelation matrix of the noise, $\mathbf{e}$.

Assuming that $\mathbf{e}$ obeys an uncorrelated process with zero mean and constant variance σ^2, this equation can be rewritten as:

$$\mathsf{MMSE} = \sigma^2 \cdot Tr\left\{(\mathbf{A}^T \cdot \mathbf{A})^{-1}\right\}$$

It applies to the determinate case. In the overdeterminate case one should use:

$$\mathsf{MMSE} = \sigma^2 \cdot Tr\left\{\mathbf{A}^{+} \cdot (\mathbf{A}^T)^{+}\right\} \qquad (7)$$

2.4 Use of the evaluation criteria

Summarizing, the evaluation criteria of importance in this study are: the maximization criterion SEL (Equation 5) for improvement of accuracy, and the minimization criteria ACC (Equation 6) and MMSE (Equation 7) for improvement of precision.

For conformity, we decided to cast ACC and MMSE into maximization criteria. Equivalent maximization criteria can be accomplished in several ways, e.g. $-$ACC and $-$MMSE, respectively, or ACC^{-1} and MMSE^{-1}, respectively.

Numerical values delivered by maximization criteria are referred to by different names in the literature – e.g. utility, quality, profit, merit, record, rate, score, fitness, etc. – depending mainly on the problem domain or search method concerned. The term "fitness" is part of GA jargon, but will nevertheless be used hereafter in descriptions concerning SA and SE as well, for further conformity.

Casting ACC and MMSE into equivalent maximization criteria does, however, not change the aforementioned fact that the subset of wavelengths considered as globally optimal, is the total set of wavelengths ($m = M$). Hence, since the solution is known in advance, search based on ACC or MMSE alone, as opposed to search based on SEL alone, is not particularly challenging.

More importantly, though, in view of the aforementioned trade-off between accuracy and precision, combined criteria such as e.g. SEL/ACC and SEL/MMSE seem better choices from an analytical-chemical point of view. However, combined criteria complicate the search and thus would unduly interfere with our main purpose to better understand the search methods on a comparative basis.

Based on the above considerations, the following evaluation criteria were chosen in this study:

- SEL;
- $(m\,\mathsf{ACC})^{-1}$;
- $(m\,\mathsf{MMSE})^{-1}$.

(In the latter two cases, the multiplication by m is to ensure that optimal subsets of wavelengths may comprise $m < M$ wavelengths.)

3 Experimental

This section briefly describes the search methods used for wavelength selection in this study, and the method validation criteria used for their mutual comparison.

For the comparison to be meaningful, several conditions are kept invariant with respect to the methods. These are discussed first.

3.1 Invariant conditions

Invariant conditions in this comparative study are: the data set, the representation of candidate subsets of wavelengths, the computer language in which the methods were programmed, the computer hardware platform on which the programs for the methods were run, and how "the solution" obtained after a run is defined.

3.1.1 Data set

The data set used in this study is a set of absorption coefficients ($a_{i,j}$) for the four RNA nucleotides A, C, G, and U ($n = 4$), measured at 36 equidistant UV-wavelengths ($M = 36$), and a spectrum ($\mathbf{d}$) of a mixture of these compounds, measured at

these wavelenths. This data set – hereafter called `RNA.dat` – was obtained from the literature [41] and has been analyzed in numerous other studies as well.

The choice of `RNA.dat` is primarily motivated by the fact that it is accompanied by foreknowledge which can be used for verification of results obtained by the search methods. (Verification is considered important at the present stage wherein still much can be learned as to how the search methods, especially the GA, perform in wavelength selection.) The following two avenues for verification are available.

Firstly, since `RNA.dat` is comparatively small ($M = 36$), it is possible to find globally optimal subsets of wavelengths through enumerative search (Section 1.2 above) within less than a couple of days of computation on a fast computer. In this way, the GA, SA, and SE can be judged as to whether they manage to find the global optimum.

Secondly, since the mixture was created from known amounts of components, the true concentrations, c_j, of these components are known. In this way, the concentrations estimated from any subset of wavelengths (using Equation 2) can be compared with the true concentrations. The true concentrations are [41]:

c_1 : 0.0000144 moles A/liter;
c_2 : 0.0000155 moles C/liter;
c_3 : 0.0000151 moles G/liter;
c_4 : 0.0000154 moles U/liter.

3.1.2 Representation

Candidate subsets in any of the methods used are represented, or encoded, as binary strings, or *bitstrings*. Each bitstring comprises M bits that mask the M wavelengths in the spectrum: bit value "1" stands for "select the corresponding wavelength", whereas bit value "0" stands for "do not select the corresponding wavelength". An example of this so-called *binary subset encoding* (see Chapter 2 (Section 2.2) in this thesis) is the following 36-bit bitstring:

101010101010101010101010101010101010

which proposes the selection of all odd-indexed wavelengths among 36 wavelengths.

It is important to realize that binary subset encoding, as described above for the purpose of wavelength selection, is constrained in that the number of 1-bits – or, effectively, the number of selected wavelengths – may not be smaller than n (since otherwise Equation 1 can not be solved). How this encoding constraint is observed by the methods used, follows from their description given below.

3.1.3 Software

All software needed for this study was programmed in the computer language C (ANSI standard). The SA and SE methods were programmed from scratch. The GA was programmed using the software library GATES [28, 29], comprising domain-independent routines (written in C, ANSI standard).

The aforementioned SVD routine for computation of the generalized inverse (Section 2.1 above), was adopted from [37] (with minor adaptations to achieve a better interfacing with other routines).

3.1.4 Hardware

The programs for the methods were run on a SUN computer (Sparc Station 2), under the UNIX operating system.

3.1.5 Definition of the solution

The solution found in a run by any of the methods, is defined as: the best candidate subset of wavelengths encountered during the run.

3.2 Methods

This section briefly describes the GA, SA, and SE as they were implemented for the purposes of this study.

3.2.1 GA

GAs comprise a large family of non-local search techniques [11, 9, 5, 4, 35, 12, 20, 30, 31] which employ a probabilistic search heuristic based loosely upon the principles of natural evolution according to Darwin [3]; for a detailed general treatise, the reader is referred to Chapters 1 and 2 in this thesis. Recently, GAs have been applied to a variety of subset selection problems [40, 27, 24, 23, 26], including wavelength selection problems in UV and NIR spectrometry.

In general, GAs maintain a population of N strings which represent candidate solutions that are in a constant competition for survival. Using generalized evolutionary operators such as selection, recombination, and mutation, the population evolves dynamically towards improvement in a number of time steps, or "generations". Thereby, N (a control parameter) remains constant; a suitable value for N often lies between 50 and 500.

Our GA operates as follows.

First, all bits in the N bitstrings comprising the population are set to a random binary value (0 or 1). In this form, the population enters the evolution cycle, which is briefly discussed next; for more details, the reader is referred to Chapter 2 (Section 3) in this thesis.

The first step in the evolution cycle involves decoding the bitstrings, and to evaluate the thus obtained candidate subsets of wavelengths using the evaluation criterion of interest (Section 2.4 above). In doing so, the fitnesses of the bitstrings are obtained.

The second step in the evolution cycle involves scaling the fitnesses in a way considered appropriate for the selection criterion used in the next step: reproduction. The fitness scaling proceeds in two consecutive steps: (1) linear scaling, and (2) normalization; for further details, the reader is referred to Chapter 2 (Section 6.1) in this thesis.

The third step in the evolution cycle is reproduction: N copies are made of N strings indicated, or "selected", in the population. The strings are selected probabilistically with an expected rate proportional to their scaled fitness; this selection criterion has become known as roulette selection. Accordingly, the new set of N bitstrings, called the temporary population, may be expected to feature a higher fitness, on average. (Note: the selection criterion mentioned here performs selection of bitstrings, i.e. not of wavelengths.)

The fourth step in the evolution cycle is a random pairing of the bitstrings in the temporary population. Pairing serves the next step: recombination.

The fifth step in the evolution cycle is the application of a recombination procedure: a recombination operator is applied to all pairs of bitstrings assigned in the preceding step. The bitstrings in each pair of bitstrings exchange fractions in a way such that the loss of implicit information relevant for the search task is minimal; this principle is generally referred to as the *building block principle*. The recombination operator in the GA used here is **B_MX**, discussed in Chapter 2 (Section 5.2.2) in this thesis.

The sixth step in the evolution cycle is the application of a mutation procedure: a mutation operator is applied to all bitstrings in the temporary population in order to invert the value of a small fraction of randomly selected bits. This serves to ensure that all parts of the search space remain reachable for the recombination operator, especially in the later stages of the search when population diversity is lost. The mutation operator in the GA used here is **B_M**, discussed in Chapter 2 (Section 5.2.3) in this thesis.

The seventh step closes the evolution cycle: the population is replaced by the temporary population.

The following remarks on the above GA scheme scheme are in place:

1. Related to the probabilistic nature of the GA, a termination criterion needs to be predefined, which, if it succeeds, stops execution; the best-so-far solution is then presented to the user. In our case, the termination criterion is simple: it succeeds after a predetermined target number of generations, chosen amply beyond the approximate point where one expects convergence to set in; this point can be found by empirically inspecting some preliminary runs.

2. Strickly speaking, the bitstring modification operators – **B_MX** and **B_M** – are not compatible with aforementioned encoding constraint, because they do not include provisions against bitstring modifications that result in a number of 1-bits smaller than n. However, the rate by which illegal bitstrings are incidentally produced is extremely low. This may be appreciated intuitively by addressing the following question (assuming data set `RNA.dat`, i.e. $M = 36$ and $n = 4$): what is the probability by which the number of 1-bits that occur among 36 random bit values is less than 4? The answer is: $1 + 36 + 630 + 7140$ (the number of subsets comprising 0, 1, 2 or 3 items that can be drawn from 36 items) divided by 2^{36} (the total number of subsets that can be drawn from 36

items) – i.e. approximately $1.1 \cdot 10^{-7}$. Of course, this is merely a rough estimate, as the GA samples the search space probabilistically rather than randomly. Nonetheless, it was empirically established that the rate at which the GA violates the encoding constraint as it searches, is indeed extremely low; on the rare occasions that an illegal bitstring is incidentally produced, it is simply "lethalized" (i.e. assigned zero fitness).

3. Whether the recombination operator **B_MX** complies with the building block principle, depends on whether the sequence of M bits comprising a bitstring is chosen such that bits which correlate strongly/weakly (in the sense of contributions to the bitstring's fitness) are placed at close/remote distance in the bitstring. A sequence for which this is the case, is called a *connected sequence* of bits: for details, the reader is referred to Chapter 4 (Section 4.1.2, Frame 2) in this thesis.

To make an educated guess at a connected sequence of bits is, however, difficult in complex wavelength selection problems. In this case, one is forced to choose an *ad hoc* sequence of bits, since empirically testing **B_MX** for all ($M!$) possible sequences of bits is practically not feasible.

In our case (`RNA.dat`), too, no effort was made to find an optimal sequence of bits; the order of the 36 bits comprising a bitstring was simply chosen according to the numerical order of the corresponding wavelengths as they occur in the spectrum. We briefly examined the implications of this choice by checking whether the recombination operator **B_UX** – discussed in Chapter 2 (Section 5.2.2) in this thesis – would perform better. (The performance of **B_UX** is independent of the sequence in which the bits appear in a bitstring; this operator may be considered a better choice when a high proportion of the bits are correlated.) It turned out that **B_UX** and **B_MX** perform comparably, hence there was no apparent need to switch to **B_UX**.

Configuration. After a fair amount of hand optimization based on general guidelines in the literature [9, 10] (and on our past experience with GAs), roughly the best configuration was found to be:

N	: ?
Maximum generation	: 500
Selective reproduction	: fitness-proportional
Fitness scaling mode	: linear
Sensitivity	: 6 0
Recombination mode	: **B_MX** (2 breakpoints)
p_r	: ?
Mutation mode	: **B_M**
p_m	: ?

(For a more detailed explanation of the configurational terminology, the reader is referred to Chapters 2 and 4 in this thesis.) As indicated by ?-marks, the entries for the parameters N, p_r, and p_m are variable. In this way, an experimental design involving these parameters can be conducted, thus taking into account the empirical fact that optimal GA configuration generally depends to some extent on the problem domain implied by the evaluation criterion used; it is expected that these parameters substantially affect performance. A central composite design experimental design was chosen [34]. Results obtained in using this design are shown in Section 4.1 below.

(It should be emphasized that the experimental design was only used in order to obtain a configuration that is likely to result in finding the global optimum; it was not used to fit the responses according to an assumed model.)

3.2.2 SA

SA comprises a family of non-local optimization methods, which employ probabilistic search heuristics based on principles of physical cooling of a solid [1, 21, 19]. More precisely, SA is based on the analogy between the simulation of the annealing of a solid and the problem of solving a large optimization problem. In its original meaning, annealing is the process of heating a solid and then cooling it slowly so as to remove strain and crystal imperfections. In this process, the free energy of the solid is minimized. The initial heating is necessary to avoid that the system gets trapped in a local minimum. Virtually every well-behaved function can be viewed as the free energy of some system, hence studying and imitating how nature reaches a minimum through an annealing process, potentially yields optimizing algorithms.

An annealing process, either real or simulated (SA), can be modeled as a so-called Markov process: a sequence of states that are interconnected by state transitions at consecutive time steps; in simulated annealing applied to the wavelength se-

lection problem, the states are candidate subsets of wavelengths.

In real annealing, the transition probability P_{vw} between consecutive states v and w is given by:

$$P_{vw} = \begin{cases} \exp\left(\frac{-(\mathcal{E}_w - \mathcal{E}_v)}{kT}\right) & \text{if } \mathcal{E}_w > \mathcal{E}_v \\ 1 & \text{otherwise} \end{cases}$$

where $\exp(-(\mathcal{E}_v - \mathcal{E}_w)/(kT))$ is the so-called Metropolis factor; $\mathcal{E}$ is free energy (which must be minimized), k is a constant (the Boltzmann constant), and T is temperature. Note that trials for transitions to lower energy ($\mathcal{E}_w < \mathcal{E}_v$) are always accepted. In contrast, trials for transitions to higher energy ($\mathcal{E}_w > \mathcal{E}_v$) are accepted only probabilistically: upon failure, the trial is rejected and a new transition is tried; upon success, the unfavorable state is accepted, thus enabling escape of local minima.

In SA, the transition probability P_{vw} between consecutive states v and w is given by:

$$P_{vw} = \begin{cases} \exp\left(\frac{\mathcal{F}_w - \mathcal{F}_v}{\tau}\right) & \text{if } \mathcal{F}_w < \mathcal{F}_v \\ 1 & \text{otherwise} \end{cases}$$

where $\exp((\mathcal{F}_w - \mathcal{F}_v)/\tau)$ is an equivalent of the Metropolis factor; $\mathcal{F}$ is fitness (which must be maximized), τ is a control parameter (an equivalent of temperature). Note that trials for transitions to higher fitness ($\mathcal{F}_w > \mathcal{F}_v$) are always accepted. In contrast, trials for transitions to lower fitness ($\mathcal{F}_w < \mathcal{F}_v$) are accepted only probabilistically: upon failure, the trial is rejected and a new transition is tried; upon success, the unfavorable state is accepted, thus enabling escape of local maxima; this is, however, less likely to happen when τ is decreased.

An important diagnostic parameter in SA is the so-called accepts/trials ratio, hereafter abbreviated as A/T; it is of importance in optimizing SA configuration.

In practice, SA is usually conducted as follows [22]. The search is subdivided into a number of consecutive chains. These so-called Markov chains are equally sized, i.e. they comprise the same number of states. The length of the Markov chains and their number, are control parameters. According to an empirical rule of thumb [21], the initial value of τ should be such that A/T lies approximately between 0.7 and 1.0. In going from one Markov chain to the next, τ is decreased by a small amount; this results in a lower expected A/T, thus forcing the search to converge. Many functions are conceivable for the decrement in τ, as a function of the serial index of the next Markov chain; in many cases, including ours, an exponentially decaying function is chosen. The SA is terminated when A/T approaches 0.0. Note that in SA (as described here), like in the GA (as described above), the termination criterion is based on empirical observations.

The extent to which one state is perturbed for a transition to the next state, is called the step size – another control parameter in SA. In our case, the first state is a random M-bit bitstring, and each consecutive transition from one bitstring to the next, is performed by flipping a predetermined number of randomly selected bits in the bitstring; thus, the step size is measured as a Hamming distance. Note that the bitstrings are modified in a way similar to how the mutation operator **B M** in the GA modifies bitstrings. However, like in the GA, the rate by which illegal bitstrings are incidentally produced is empirically found to be extremely low.

Recently Kalivas introduced an alternative form of the simulated annealing algorithm in analytical chemistry, which he called generalized simulated annealing (GSA) [16]. In GSA, the Metropolis factor has been modified to:

$$P_{vw} = \exp\left(\frac{\tau(\mathcal{F}_v - \mathcal{F}_w)}{\mathcal{F}_v - \mathcal{F}_o}\right)$$

where F_o is the optimal fitness value. Evidently, here a difficulty lies in the fact that the optimal fitness value has to be known beforehand. (In part, a way around this problem is to adjust this parameter during a run, i.e. to replace it by the best-so-far fitness value.) In our study, all experiments involving SA were repeated using GSA, but as the results were generally not any better, they are not presented in this paper.

Configuration. After a fair amount of hand optimization based on general guidelines in the literature [21], roughly the best configuration was found to be:

Length of Markov chains	: 100,000
Number of Markov chains	: 40
Step size	: 5 bits

Because of the analogy with cooling a solid, in SA a *specific heat* parameter can be defined. It is defined as the average fitness observed in a Markov

chain, divided by the decrement in τ after that Markov chain is completed. When a solid is slowly cooled, the specific heat suddenly increases when a phase transition takes place. Hence, finding a value of τ at which the specific heat suddenly increases, should indicate that an important change has taken place. (This can also be deduced from other factors that are monitored during a run. For instance, at a phase transition, also the average fitness observed in a Markov chain should increase, while the standard deviation thereof should decrease.) It follows from the theory of SA [21] that one is likely to find the global optimum at $\tau = \tau_o$ – the value of τ where the specific heat is maximal, i.e. where a "phase transition" takes place. Once τ_0 has been found, a new SA run – involving a *single, longer* Markov chain at $\tau = \tau_o$ – is performed to search for the optimal solution; this Markov chain is longer by a factor $(A/T)^{-1}$ at $\tau = \tau_o$.

3.2.3 SE

SE is conducted as follows. Starting from a bitstring comprised of 1-bits only (i.e. $m = M$, all wavelengths selected), in each among $M - n$ iterations that follow, a 1-bit is replaced by a 0-bit; the 1-bit is selected according the largest gain in fitness observed upon the replacement. Thus, when SE terminates, the number of 1-bits in the bitstring has been reduced to n.

Evidently, SE is deterministic (non-probabilistic). Also, SE automatically complies with aforementioned encoding constraint, as, evidently, the number of 1-bits in the string never becomes less than n during the search.

Configuration. SE can not be configured, since it is free of control parameters.

3.3 Method validation criteria

This section describes the method validation criteria used in this study: HIT (hitting the global optimum in a run), PE (percentage error in the concentration estimate of a component), and RSD (relative standard deviation in the concentration estimate of a component). These criteria have in common that they judge search methods according to the quality of the solution found, as pointed out in Section 1.4 above.

It is important to realize that no definite conclusions can be drawn from validations of the GA and SA. This may be ascribed to two sources of error. The first source of error lies in the fact that the GA and SA are probabilistic; hence, since only a finite number of runs can be carried out, statistical uncertainty in the outcoe always exists. The second source of error resides in the fact that the optimal configuration of the GA and SA is generally unknown, which should be ascribed to the huge and complex configuration space; at best, the comparison between the GA and SA can be made fair by spending an equal amount of effective effort in empirically optimizing their configuration.

3.3.1 Hitting the global optimum

HIT is a boolean method validation criterion, defined as the answer to the question: has the globally optimal subset of wavelengths – known in advance by way of enumerative search – been "hit" (i.e. found) in a run performed by the search method concerned? HIT is considered the most important method validation criterion in this study.

3.3.2 Percentage error and relative standard deviation

Both PE and RSD compare a component concentration, estimated from a particular subset of wavelengths (using Equation 2), with the true component concentration; PE is a measure of the accuracy in this estimate, whereas RSD is a measure of the precision in this estimate.

PE is defined in a straightforward way as:

$$\mathrm{PE} = \frac{| c_j - \hat{c}_j |}{c_j} \cdot 100\%$$

where: c_j is the true concentration of component j (see Section 3.1.1); $\hat{c}_j$ is the estimated concentration of component j, for a particular subset of wavelengths.

RSD is defined as [41]:

$$\mathrm{RSD} = \frac{s_{c_j}}{\|\bar{c}_j\|} \cdot 100\%$$

where $\bar{c}_j$ (mean) and s_{c_j} (standard deviation) are obtained as follows:

1. Solve Equation 1, and store the value thereby obtained for $\mathbf{c}$ (Equation 2);

2. Form $\mathbf{A}'$ (a new value for $\mathbf{A}$) and $\mathbf{d}'$ (a new value for $\mathbf{d}$) by setting:

$$a'_{ij} = a_{ij} + \frac{\epsilon}{100} \cdot u \cdot a_{ij}$$
$$d'_j = d_j + \frac{\epsilon}{100} \cdot u \cdot d_j$$

where:

$\epsilon = \epsilon_a = \epsilon_d = 2\%$ (the RSD in a_{ij} and d_j, see Section 2.2.2 above); u is a standard normally distributed random variable (mean 0 and standard deviation 1);

3. Solve Equation 1 using $\mathbf{A} = \mathbf{A}'$ and $\mathbf{d} = \mathbf{d}'$, and store the new value, $\mathbf{c}'$, thereby obtained for $\mathbf{c}$;

4. Repeat steps 2 and 3 many times (here 100) and then go to step 5;

5. Using the sequence of values $\mathbf{c}'$ for $\mathbf{c}$ thus obtained, calculate the mean and standard deviation of each c_j – that is, $\bar{c}_j$ and s_{c_j}, respectively.

From the above described procedure, it becomes clear that RSD is a measure of the instability of Equation 1, i.e. a measure of the susceptibility to changes in the solution of the system caused by small changes in the absorption coefficients, a_{ij}. It is assumed that the RSD is larger than the RSD in a_{ij} and d_j, hence Equation 1 model will amplify the original error in the data (a_{ij}).

4 Results and discussion

This section discusses the results obtained in the experimental strategy implied by the foregoing.

4.1 Using selectivity after Lorber

Based on the evaluation criterion SEL, first the global solution is determined; then, the solutions found by SE, the GA, and SA, respectively, are presented and commented on; finally, concentration estimates derived from the global solution are examined.

4.1.1 The global solution

Enumerative search was conducted in order to find the globally optimal subset of wavelengths according to SEL; results are listed in Table 1. In this way, it became possible to validate the GA, SA, and SE, as to whether they succeed in HITting the SEL-optimal subset.

Table 1: Enumerative search using SEL. The boldfaced line indicates the global optimum.

m	SEL	Subset
4	0.2362	1/11/29/36
5	**0.2402**	**1/11/29/35/36**
6	0.2387	1/3/12/29/35/36
7	0.2372	1/2/12/28/29/35/36

Table 2: SE using SEL.

m	SEL	Subset
4	0.2355	1/12/28/36
5	0.2367	1/12/28/35/36
6	0.2373	1/2/12/28/35/36
7	0.2372	1/2/12/28/29/35/36

4.1.2 SE

The results of SE using SEL are listed in Table 2.

Although the global solution was not HIT, the solution found (for $m = 6$) is near-optimal and therefore probably adequate, in practice. This may be considered an important result, since SE conducts only nearly 660 evaluations in the search space [27], which compares very favorably with the formal size of the search space ($\approx 2^{36}$, or $\approx 6.87 \cdot 10^{10}$). However, it should be borne in mind that since SE employs a local search heuristic, the fitness of the solution found, may drop drastically when more complex, larger data sets are concerned; SE seems an unstable strategy.

4.1.3 GA

The results of the GA using SEL, obtained in 15 experiments according to a three-factor central composite design, are listed in Table 3.

It can be seen that the global solution was HIT for two configurations (experiments #10 and #13). Upon repeating the experiments a number of times (for different random initial populations), it was observed that these configurations sometimes fail to HITthe global solution, whereas other configurations succeed in HITting the global solution. This underscores two important aspects of robustness that are typical of GAs in general

Table 3: GA using SEL, in 15 experiments according to a three-factor central composite design Bold-faced lines indicate the global optimum.

#	N	p_r	p_m	SEL	Subset
1	75	0.7	0.005	0.2383	1/3/12/28/35/36
2	75	0 9	0.005	0.2349	1/3/12/28/35
3	125	0.7	0 005	0.2313	1/2/11/28/36
4	125	0.9	0 005	0.2373	1/2/12/28/35/36
5	75	0.7	0.015	0.2296	1-3/13/38/29/32/35/36
6	75	0.9	0.015	0.2373	1/2/12/28/35/36
7	125	0.7	0.015	0.2372	1/2/12/28/29/35/36
8	125	0.9	0.015	0.2281	1-3/11/12/28/29/32/35/36
9	150	0.8	0 01	0.2221	1/2/12/28/29/32/35
10	**100**	**0.8**	**0.01**	**0.2402**	**1/11/29/35/36**
11	50	0 8	0 01	0.2022	1-9/12/13/27-29/31-34
12	100	0.8	0.0001	0.2383	1/3/12/28/35/36
13	**100**	**0.8**	**0.02**	**0.2402**	**1/11/29/35/36**
14	100	0.6	0 01	0.2402	1/11/29/35/36
15	100	1 0	0 01	0 2349	1/3/12/28/35

[31]: (1) GAs are not easily misled by sub-optima in the search space; (2) GA performance is not dramatically affected by small changes in configuration.

In order to obtain a quantitative indication of the first type of robustness, experiment #10 was repeated ten times: the average fitness of the respective solutions was 0.2381, with a standard deviation of 0.001.

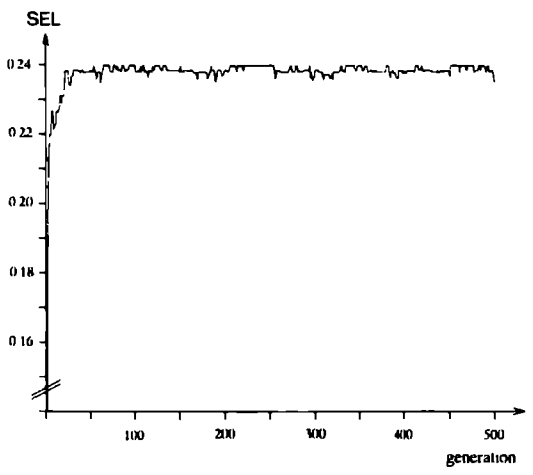

Figure 1: Evolution of SEL for the best bitstring in the population, in a run with $N = 150$, $p_r = 0.8$, and $p_m = 0.01$.

For an arbitrary run according to experiment #9, Figure 1 shows the evolution of SEL for the best bitstring in the population. From this figure it can be seen that (near-)optimal solutions are found after approximately 80 generations, i.e. the GA conducts approximately 12,000 evaluations in the search space (since $N = 150$) Although this number is almost 2 orders of magnitude larger than the number of evaluations in SE, it still compares favorably with the formal size of the search space. More importantly, though the running time is still acceptable (approximately 20 minutes real-time, using aforementioned hardware).

4.1.4 SA

The results of SA using SEL are listed in Table 4

It can be seen that the global solution was not HIT, regardless of the fact that SA conducts by far the largest number of evaluations in the search space. Moreover, the solution found by SA is inferior to the near-optimal solution found by SE.

Similar results were obtained in runs at other r values for which a high specific heat was observed

4.1.5 Concentration estimation

The accuracy (PE) and precision (RSD) in the estimates of the component concentrations, for both the entire set of wavelengths (as a reference) and the SEL-optimal subset of wavelengths, are listed in Tables 5 and 6, respectively.

Table 4: SA using SEL. The bold-faced line indicates maximal specific heat.

τ	A/T	Specific heat	SEL Average	SEL St. dev.
0.2	0.940	0.0112	0.1164	0 0213
0.1	0.889	0.0434	0.1187	0.0206
0.08	0.860	0.062	0.1197	0.0202
0.06	0.822	0.1067	0.1212	0.0197
0.05	0.772	0.1570	0.1168	0.0195
0.04	0.738	0.2200	0.1246	0.0189
0.03	0.654	0.3807	0.1274	0.0186
0.02	0.502	0.9057	0.1337	0.0193
0 019	0.422	1.2689	0.1256	0.0205
0.018	0.427	1.2820	0.1269	0.0197
0.017	0.366	1.4989	0.1279	0.0210
0.016	0.350	1.7206	0.1308	0.0214
0.015	0.297	1.8721	0.1319	0.0233
0 014	0.272	3.925	0.1370	0.0259
0.013	0.198	3.4082	0.1398	0.0261
0 012	0.202	4.0156	0.1408	0.0249
0.0115	0.165	3.9722	0.1404	0.0233
0.011	0.136	5.2446	0.1440	0.0228
0.0105	0.125	4.1083	0.1442	0.0292
0.01	0 149	6.1109	0.1564	0.0231
0.0099	0.095	4.3903	0.1480	0.0225
0.0098	0.081	5.5977	0.15508	0.0277
0.0097	0 094	4.9859	0.1481	0.0231
0.0096	**0.072**	**12.9099**	**0.1607**	**0.0322**
0.0095	0.107	5.6549	0.1592	0.0220
0.0094	0 085	6.9519	0.1507	0.0239
0 0093	0.073	7.638	0.1556	0.0263
0.0092	0 040	0	0.1839	0.0314
0.0091	0.040	10.6744	0.1784	0.0327
0.009	0.050	11.2034	0.1673	0.0277
0.0089	0.050	5.4951	0.1634	0.0271
0.0088	0 051	5.4488	0.1544	0.0214
0.0087	0.053	7.3825	0.1565	0.0226
0.0086	0.029	8 9277	0 1749	0.0289
0.0085	0.034	6.8040	0.1762	0.0238
0.0084	0.027	6.4091	0.1517	0.0327
0 0083	0.034	7.6949	0.1713	0.0211
0.0082	0.027	9.2461	0.1724	0.0255
0.008	0 053	6.858	0.1657	0.0225
0 007	0.034	7.9346	0.1823	0.0261
New run for $\frac{100.000}{0\ 072}$ iterations at $\tau = 0.0096$. Subset: 1/3/7/14/28/34-36; SEL: 0.2215				

From these tables, the following general tendencies can be inferred. The selection of wavelengths (rather than using the entire set of wavelengths) leads to a much better accuracy in the concentration estimates for components 1 (A) and 4 (U), but also to a much worse precision in the concentration estimates for component 4.

Table 5: PE for all wavelengths, and PE for SEL-optimal subset (1/11/29/35/36, Table 1).

j	PE for all wavelengths	PE for SEL-optimal subset
1 (A)	31.1	0.7
2 (C)	6.1	6.6
3 (G)	6.0	6.6
4 (U)	35.1	1.3

Table 6: RSD for all wavelengths, and RSD for SEL-optimal subset (1/11/29/35/36, Table 1).

j	RSD for all wavelengths	RSD for SEL-optimal subset
1 (A)	27.4	23.6
2 (C)	3.1	5.2
3 (G)	6.6	11.1
4 (U)	17.4	35.2

Similar general tendencies (relative to using the entire set of wavelengths) were observed for the subsets of wavelengths found by the GA, SA, and SE, respectively; which of these methods performs better, based upon validation by PE or RSD, could not be decided.

4.2 Using accuracy after Lorber

Based on the evaluation criterion $(m\,\mathrm{ACC})^{-1}$, first the global solution is determined; then, the solutions found by SE, the GA, and SA, respectively, are presented and commented on; finally, concentration estimates derived from the global solution are examined.

Comments are limited to trends not mentioned in Section 4.1 above.

4.2.1 The global solution

Enumerative search was conducted in order to find the globally optimal subset of wavelengths according to $(m\,\mathrm{ACC})^{-1}$; results are listed in Table 7.

4.2.2 SE

The results of SE using $(m\,\mathrm{ACC})^{-1}$ are listed in Table 8.

It can be seen that the global solution was HIT.

Table 7: Enumerative search using $(m\,\mathsf{ACC})^{-1}$. The bold-faced line indicates the global optimum.

m	$(m\,\mathsf{ACC})^{-1}$	Subset
4	**0.0164**	**1/15/28/35**
5	0.0147	1/15/27/28/35
6	0.0135	1/2/15/27/28/35

Table 8: SE using $(m\,\mathsf{ACC})^{-1}$. The bold-faced line indicates the global optimum.

m	$(m\,\mathsf{ACC})^{-1}$	Subset
4	**0.0164**	**1/15/28/35**
5	0.0147	1/15/27/28/35
6	0.0135	1/2/15/27/28/35

4.2.3 GA

The results of the GA using $(m\,\mathsf{ACC})^{-1}$, obtained in 15 experiments according to a three-factor central composite design, are listed in Table 9.

It can be seen that the global solution was HIT for three configurations (experiments #6, #10, and #11).

For an arbitrary run according to experiment #9, Figure 2 shows the evolution of $(m\,\mathsf{ACC})^{-1}$ for the best bitstring in the population.

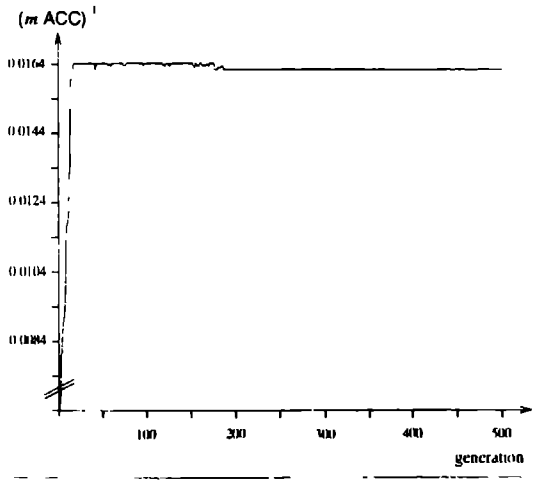

Figure 2: Evolution of $(m\,\mathsf{ACC})^{-1}$ for the best bitstring in the population, in a GA-run with $N = 150$, $p_r = 0.8$, and $p_m = 0.01$.

4.2.4 SA

The results of SA using $(m\,\mathsf{ACC})^{-1}$ are listed in Table 10.

It can be seen that the global solution was not HIT.

4.2.5 Concentration estimation

The accuracy (PE) and precision (RSD) in the estimates of the component concentrations, for both the entire set of wavelengths (as a reference) and the $(m\,\mathsf{ACC})^{-1}$-optimal subset of wavelengths, are listed in Tables 11 and 12, respectively.

4.3 Using minimum mean squared error after Sasaki

Based on the evaluation criterion $(m\,\mathsf{MMSE})^{-1}$, first the global solution is determined; then, the solutions found by SE, the GA, and SA, respectively, are presented and commented on; finally, concentration estimates derived from the global solution are examined.

Comments are limited to trends not mentioned in Section 4.1 above.

4.3.1 The global solution

Enumerative search was conducted in order to find the globally optimal subset of wavelengths according to $(m\,\mathsf{MMSE})^{-1}$; results are listed in Table 13.

4.3.2 SE

The results of SE using $(m\,\mathsf{MMSE})^{-1}$ are listed in Table 14.

Although the global solution was not HIT, the solution found (for $m = 5$) is near-optimal.

4.3.3 GA

The results of the GA using $(m\,\mathsf{MMSE})^{-1}$, obtained in 15 experiments according to a three-factor central composite design, are listed in Table 15.

It can be seen that the global solution was HIT for all configurations, except two (experiments #5 and #14).

For an arbitrary run according to experiment #9, Figure 3 shows the evolution of

Table 9: GA using $(m\,\mathrm{ACC})^{-1}$, in 15 experiments according to a three-factor central composite design. Bold-faced lines indicate the global optimum.

#	N	p_r	p_m	$(m\,\mathrm{ACC})^{-1}$	Subset
1	75	0.7	0.005	0.0162	1/16/27/35
2	75	0.9	0.005	0.0163	1/15/27/35
3	125	0.7	0.005	0.0163	1/15/27/35
4	125	0.9	0.005	0.0161	1/15/27/34
5	75	0.7	0.015	0.0161	1/15/27/34
6	**75**	**0.9**	**0.015**	**0.0164**	**1/15/28/35**
7	125	0.7	0.015	0.0161	1/15/27/34
8	125	0.9	0.015	0.0161	1/15/27/34
9	150	0.8	0.01	0.0161	1/15/27/34
10	**100**	**0.8**	**0.01**	**0.0164**	**1/15/28/35**
11	**50**	**0.8**	**0.01**	**0.0164**	**1/15/28/35**
12	100	0.8	0.0001	0.0162	1/14/27/35
13	100	0.8	0.02	0.0163	1/15/27/35
14	100	0.6	0.01	0.0163	1/16/28/35
15	100	1.0	0.01	0.0161	1/15/27/34

$(m\,\mathrm{MMSE})^{-1}$ for the best bitstring in the population.

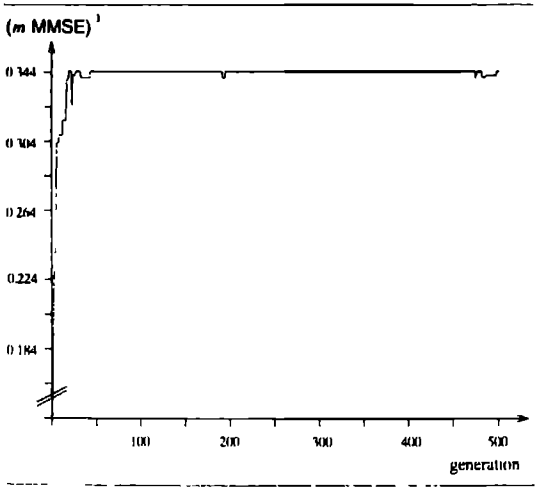

Figure 3: Evolution of $(m\,\mathrm{MMSE})^{-1}$ for the best bitstring in the population, in a GA-run with $N = 150$, $p_r = 0.8$, and $p_m = 0.01$.

4.3.4 SA

The results of SA using $(m\,\mathrm{ACC})^{-1}$ are listed in Table 16.

It can be seen that the global solution was not HIT.

4.3.5 Concentration estimation

The accuracy (PE) and precision (RSD) in the estimates of the component concentrations, for both the entire set of wavelengths (as a reference) and the $(m\,\mathrm{MMSE})^{-1}$-optimal subset of wavelengths. are listed in Tables 17 and 18, respectively.

Table 10: SA using $(m\,\mathsf{ACC})^{-1}$. The bold-faced line indicates maximal specific heat.

τ	A/T	Specific heat	$(m\,\mathsf{ACC})^{-1}$ Average	St. dev.
0.005	0.803	0.1432	0.00361	0.00189
0.0045	0.784	0.1579	0.00376	0.00183
0.004	0.763	0.1928	0.00380	0.00177
0.0038	0.755	0.2204	0.00379	0.00177
0.0035	0.737	0.2340	0.00389	0.00173
0.0032	0.713	0.2819	0.00393	0.00168
0.003	0.710	0.3061	0.00390	0.00165
0.0029	0.691	0.2995	0.00394	0.00162
0.0028	0.687	0.3377	0.00396	0.00163
0.0027	0.681	0.3697	0.00400	0.00157
0.0026	0.674	0.3364	0.00402	0.00155
0.0025	0.657	0.3628	0.00404	0.00151
0.0024	0.653	0.4211	0.00402	0.00154
0.0023	0.637	0.4297	0.00410	0.00145
0.0022	0.623	0.4444	0.00413	0.00146
0.0021	0.614	0.4951	0.00414	0.00144
0.002	0.601	0.5594	0.00413	0.00143
0.0019	0.579	0.5231	0 00425	0.00137
0.0018	0.571	0.6229	0.00420	0.00137
0.0017	0.548	0.5745	0.00426	0.00132
0.0016	0.537	0.6155	0.00427	0.00125
0.0015	0.506	0.6756	0.00436	0.00123
0 0014	0.489	0.7283	0.00438	0.00121
0.0013	0.441	0.7950	0.00458	0.00141
0.0012	0.412	1.1109	0.00454	0.00121
0.0011	0.391	1.1435	0.00459	0.00117
0.001	0.355	1.1647	0.00465	0.00108
0.0009	0.304	1.5856	0.00478	0.00109
0.0008	0.256	2.0708	0.00497	0.00116
0.0007	0.178	3.4543	0.00530	0.00126
0.0006	0.129	3.5515	0.00553	0.00126
0.0005	**0.063**	**8.9944**	**0.00636**	**0.00148**
0.0004	0.038	4.2052	0.00611	0 00119
0.0003	0.012	4.2184	0.00663	0.00067
0.0002	0.004	0	0.00787	0.00029
0.0001	0	0	0.00835	0.00101
9e-05	0	0	0.00770	0.00042
8e-05	0	0	0.00891	0.00056
7e-05	0	0	0.00956	0.00012
6e-05	0	0	0.00956	0.00011

New run for $\frac{100,000}{0.063}$ iterations at $\tau = 0.0005$:
Subset: 1/14/15/23/28/29/36;
$(m\,\mathsf{ACC})^{-1}$: 0.01082

Table 11: PE for all wavelengths, and PE for $(m\,\mathsf{ACC})^{-1}$-optimal subset (1/15/28/35, Table 7).

j	PE for all wavelengths	PE for $(m\,\mathsf{ACC})^{-1}$-optimal subset
1 (A)	31.1	0.9
2 (C)	6.1	6.6
3 (G)	6.0	5.3
4 (U)	35.1	3.8

Table 12: RSD for all wavelengths, and RSD for $(m\,\mathsf{ACC})^{-1}$-optimal subset (1/15/28/35, Table 7).

j	RSD for all wavelengths	RSD for $(m\,\mathsf{ACC})^{-1}$-optimal subset
1 (A)	27.4	30.7
2 (C)	3.1	5.6
3 (G)	6.6	12.9
4 (U)	17.4	44.7

Table 13: Enumerative search using $(m\,\mathsf{MMSE})^{-1}$. The bold-faced line indicates the global optimum.

m	$(m\,\mathsf{MMSE})^{-1}$	Subset
4	0.3325	1/15/28/35
5	**0.3442**	**1/15/27/28/35**
6	0.3408	1/2/15/27/28/35

Table 14: SE using $(m\,\mathsf{MMSE})^{-1}$.

m	$(m\,\mathsf{MMSE})^{-1}$	Subset
4	0.3275	1/18/28/35
5	0.3424	1/18/27/28/35
6	0.3377	1/2/18/27/28/35
7	0.3253	1/2/18/27/28/35/36

Table 15: GA using $(m\,\mathsf{MMSE})^{-1}$, in 15 experiments according to a three-factor central composite design. Bold-faced lines indicate the global optimum.

#	N	p_r	p_m	$(m\,\mathsf{MMSE})^{-1}$	Subset
1	**75**	**0.7**	**0.005**	**0.3442**	**1/15/27/28/35**
2	**75**	**0.9**	**0.005**	**0.3442**	**1/15/27/28/35**
3	**125**	**0.7**	**0.005**	**0.3442**	**1/15/27/28/35**
4	**125**	**0.9**	**0.005**	**0.3442**	**1/15/27/28/35**
5	75	0.7	0.015	0.3289	1/2/18/26/27/28/35
6	**75**	**0.9**	**0.015**	**0.3442**	**1/15/27/28/35**
7	**125**	**0.7**	**0.015**	**0.3442**	**1/15/27/28/35**
8	**125**	**0.9**	**0.015**	**0.3442**	**1/15/27/28/35**
9	**150**	**0.8**	**0.01**	**0.3442**	**1/15/27/28/35**
10	**100**	**0.8**	**0.01**	**0.3442**	**1/15/27/28/35**
11	**50**	**0.8**	**0.01**	**0.3442**	**1/15/27/28/35**
12	**100**	**0.8**	**0.0001**	**0.3442**	**1/15/27/28/35**
13	**100**	**0.8**	**0.02**	**0.3442**	**1/15/27/28/35**
14	100	0.6	0.01	0.3368	1/15/27/28/34
15	**100**	**1.0**	**0.01**	**0.3442**	**1/15/27/28/35**

Table 16: SA using $(m\,\mathrm{MMSE})^{-1}$. The bold-faced line indicates maximal specific heat.

τ	A/T	Specific heat	$(m\,\mathrm{MMSE})^{-1}$ Average	St. dev.
0.06	0.734	0.2140	0.1372	0.0276
0.05	0.695	0.2785	0.1393	0.0261
0.04	0.631	0.3964	0.1428	0.0251
0.035	0.589	0.4716	0.1449	0.0241
0.03	0.537	0.6268	0.1475	0.0236
0.025	0.467	0.8361	0.1511	0.0234
0.02	0.375	1.2670	0.1561	0.0226
0.018	0.329	1.5717	0.1591	0.0231
0.016	0.283	1.9882	0.1615	0.0230
0.014	0.224	3.0682	0.165	0.0234
0.012	0.147	4.1554	0.1759	0.0272
0.011	0.092	4.9576	0.1925	0.0353
0.01	0.089	5.3885	0.1832	0.0248
0.0098	0.070	6.9065	0.1902	0.0287
0.0096	0.073	9.1335	0.1855	0.0258
0.0094	0.069	9.7859	0.1882	0.0269
0.0092	0.071	5.4202	0.1827	0.0212
0.0091	0.040	5.9473	0.2068	0.0331
0.0090	0.051	8.8869	0.1931	0.0264
0.0089	0.059	8.2737	0.1874	0.0238
0.0088	0.043	8.3779	0.1986	0.0282
0.0087	0.042	6.7122	0.1940	0.0236
0.0086	0.029	8.8079	0.2068	0.0274
0.0085	0.030	9.3625	0.2057	0.0280
0.0084	0 033	9.6110	0.2027	0.0276
0.0083	0.026	7.3339	0.2061	0.0256
0.0082	0.043	6.8667	0.1892	0.0213
0.0081	0.033	10.0626	0.1961	0.0249
0.0080	0.038	7.8922	0.1916	0.0212
0.0079	0.032	6.0395	0.1948	0.0239
0.0078	0.029	9.3227	0.1982	0.0248
0.0077	0.005	0	0.2326	0.0174
0.0076	0	0	0.2417	7.435e-08
0.0075	0.002	7.3025	0.2381	0.0141
0.0074	0.011	7.4185	0.2103	0.0202
0.0073	**0.015**	**13.1857**	**0.2092**	**0.0232**
0.0072	0.016	5.8281	0.2029	0.0202
0.0071	0.003	0	0.2344	0.0187
0.007	0	0	0.2445	4.881e-07
0.0068	0	0	0.2444	4.881e-07

New run for $\frac{100,000}{0.015}$ iterations at $\tau = 0.0073$:
Subset: 1/2/10/15/27/28/35/36;
$(m\,\mathrm{MMSE})^{-1}$: 0.2867

Table 17: PE for all wavelengths, and PE for $(m\,\mathrm{MMSE})^{-1}$-optimal subset (1/15/27/28/35, Table 13).

j	PE for all wavelengths	PE for $(m\,\mathrm{MMSE})^{-1}$-optimal subset
1 (A)	31.1	6.2
2 (C)	6.1	2.5
3 (G)	6.0	2.0
4 (U)	35.1	1.6

Table 18: RSD for all wavelengths, and RSD for $(m\,\mathrm{MMSE})^{-1}$-optimal subset (1/15/27/28/35, Table 13).

j	RSD for all wavelengths	RSD for $(m\,\mathrm{MMSE})^{-1}$-optimal subset
1 (A)	27.4	28.5
2 (C)	3.1	6.1
3 (G)	6.6	11.3
4 (U)	17.4	31.5

5 Conclusions

A comparative study involving a genetic algorithm, simulated annealing, and stepwise elimination, as methods for wavelength selection in multicomponent analysis, was presented. For each of the three evaluation criteria used in this study, the genetic algorithm succeeded in finding the globally optimal solution. In contrast, simulated annealing failed in finding the globally optimal solution for each of the evaluation criteria. Stepwise elimination succeeded in finding the global optimum only for one of the evaluation criteria; for the others, near-optimal solutions were found.

No definite conclusions can be drawn from these results, because optimal configurations for genetic algorithms and simulated annealing are generally unknown; standard methodology to that end is not available. Moreover, the picture may change drastically when a more complex, larger data set is subjected to wavelength selection by the methods used in this study.

Acknowledgments

This research was carried out under the auspices of the Dutch Foundation for Chemical Research

(SON) on behalf of the Dutch Foundation for Informatics (Computing Science) Research (SION) with financial aid from the Dutch Organization for Scientific Research (NWO), Grant 700-344-007 Contributions to this work by W.A.C van der Heijden, W.G.F. van Duijnhoven, N M. Faber, and M.J.P Gerritsen, are gratefully acknowledged.

References

[1] Aarts, E H L and Korst, J H M *Simulated Annealing and Boltzmann Machines A Stochastic Approach to Combinatorial Optimization and Neural Computing* Wiley, Chichester, 1989

[2] Carry, W P , Beebe, K R , Kowalski, B R , Illman, D L , and Hirschfeld, T Selection of Adsorbates for Chemical Sensor Arrays by Pattern Recognition *Analytical Chemistry*, 58 149, 1986

[3] Darwin, C *The Origin of Species by Means of Natural Selection the Preservation of Favoured Races in the Struggle for Life* John Murray, London, 1859 First edition text, edited with an introduction by J W Burrow, published in Pinguin Books 1968

[4] Davidor, Y *Genetic Algorithms and Robotics A Heuristic Strategy for Optimization* World Scientific, Singapore, 1991

[5] Davis, L , editor *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York, NY, 1991

[6] Forsythe, G E , Malcolm, M A , and Moler, C B *Computer Methods for Mathematical Computations* Prentice Hall Inc , NJ, 1977

[7] Garey, M R and Johnson, D S *Computers and Intractability A Guide to the Theory of NP-Completeness* W H Freeman and Company, San Francisco, CA, 1979

[8] Gill, P E , Murray, W , and Wright, M H *Practical Optimization* Academic Press Inc , London, 1981

[9] Goldberg, D E *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, Reading, MA, 1989

[10] Grefenstette, J J Optimization of control parameters for genetic algorithms In *IEEE Transactions on Systems, Man, and Cybernetics*, page 122 IEEE, SMC-16(1), January/February 1986 Vol 16(1)

[11] Holland J H *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, MI, 1975 Revised print MIT Press, Cambridge, MA, 1992

[12] Holland, J H Genetic algorithms *Scientific American*, 267(1) 44–50, July 1992

[13] Juhl, L L and Kalivas, J H Evaluation of Experimental Designs for Multicomponent Determinations by Spectrophotometry *Analytica Chimica Acta*, 207 125, 1988

[14] Junker, A and Bergmann, G Auswahl, Vergleich und Bewertung optimaler Arbeitsbedingungen fur die quantitative Mehrkomponenten-Analyse 2 Mitteilung Auswahl optimaler Meßstellen mit Hilfe der Distension der Eichmatrix *Fresenius' Zeitschrift der Analytischen Chemie*, 278 191, 1976

[15] Kalivas, J H A Simplex Optimized Inductively Coupled Plasma Spectrometer with Minimization of Interferences *Applied Spectroscopy*, 41 1338, 1987

[16] Kalivas, J H Generalized Simulated Annealing for Calibration Sample Selection from an Existing Set and Orthogonalization of Undesigned Experiments *Journal of Chemometrics*, 5 37, 1991

[17] Kalivas, J H Optimization using variations of simulated annealing *Chemometrics and Intelligent Laboratory Systems*, 15 1–12, 1992

[18] Kalivas, J H , Roberts, N , and Sutter, J M Global Optimization by Simulated Annealing with Wavelength Selection for Ultraviolet-Visible Spec trophotometry *Analytical Chemistry*, 61 2024, 1989

[19] Kirkpatrick, S , Gelatt, C D Jr , and Vecchi, M P Optimization by simulated annealing *Science*, 220 671–680, 1983

[20] Koza, J R *Genetic Programming On the Programming of Computers by Means of Natural Selection* MIT Press, Cambridge, MA, 1992

[21] Laarhoven van P J M and Aarts, E H L *Simulated Annealing Theory and Applications* Mathematics and its Applications (East European Series) D Reidel, Dordrecht, 1987

[22] Laarhoven van, P J M and Aarts, E H L *Simulated Annealing Theory and Applications* D Reidel Publishing Company, 1987

[23] Leardi, R , Boggia, R , and Terrile, M Genetic algorithms as a strategy for feature selection *Journal of Chemometrics*, 6 267–281, 1992

[24] Li, T -H , Lucasius, C B , and Kateman, G Optimization of calibration data with the dynamic genetic algorithm *Analytica Chimica Acta*, 268(1) 123–134, 1992

[25] Lorber, A. Error Propagation and Figures of Merit for Quantitation by Solving Matrix Equations. *Analytical Chemistry*, 58:1167, 1986.

[26] Lucasius, C.B., Dane, A.D., and Kateman, G. On k-medoid clustering of large data sets with the aid of a genetic algorithm: Background, feasibility and comparison. *Analytica Chimica Acta*, 1993. In press.

[27] Lucasius, C.B. and Kateman, G. Genetic algorithms for large-scale optimization in chemometrics. An application. *Trends in Analytical Chemistry*, 10:254–261, 1991.

[28] Lucasius, C.B. and Kateman, G. GATES towards evolutionary large-scale optimization: A software-oriented approach to genetic algorithms. Part 1. General perspective. *Computers & Chemistry*, 1993. In press.

[29] Lucasius, C.B. and Kateman, G. GATES towards evolutionary large-scale optimization: A software-oriented approach to genetic algorithms. Part 2. Toolbox description. *Computers & Chemistry*, 1993. In press.

[30] Lucasius, C.B. and Kateman, G. Understanding and using genetic algorithms. Part 1. Concepts, properties and context. *Chemometrics and Intelligent Laboratory Systems*, 19:1–33, 1993.

[31] Lucasius, C.B. and Kateman, G. Understanding and using genetic algorithms. Part 2. Representation, configuration and hybridization. *Chemometrics and Intelligent Laboratory Systems*, 1993. In press.

[32] Maffioli, F. The complexity of combinatorial optimization problems and the challenge of heuristics. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, page Chapter 5, New York, NY, 1979. Wiley.

[33] Malinowski, E.R. Obtaining the Key Set of Typical Vectors by Factor Analysis and Subsequent Isolation of Component Spectra. *Analytica Chimica Acta*, 134:129, 1982.

[34] Massart, D.L., Vandeginste, B.G.M., Deming, S.N., Michotte, Y., and Kaufman, L. *Chemometrics: A textbook*. Elsevier, Amsterdam, 1988.

[35] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial Intelligence Series. Springer-Verlag, Berlin, 1992.

[36] Noble, N. *Applied Linear Algebra*. Prentice Hall, New York, 1969.

[37] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 1988.

[38] Salamin, P.A., Bartels, H., and Forster, P. A Wavelength and Optical Path Length Selection Procedure for Spectroscopic Multicomponent Analysis. *Chemometrics and Intelligent Laboratory Systems*, 11:57, 1991.

[39] Sasaki, K., Kawata, S., and Minami, S. Optimal Wavelength Selection for Quantitative Analysis. *Applied Spectroscopy*, 40(2):185, 1986.

[40] Siedlecki, W. and Sklansky, J. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10:335–347, 1989.

[41] Zscheile, F.P., Murray, H.C., Baker, G.A., and Peddicord, R.G. Instability of Linear Systems derived from Spectrophotometric Analysis of Multicomponents Systems. *Analytical Chemistry*, 34.1776, 1962.

Summary

The following is a summary of Chapters 1 to 13.

Part 1. Acquisition

Chapter 1 is the first part of a tutorial on genetic algorithms. The fundamental principles of genetic algorithms are outlined, as well as their most important properties and the broader methodological context they fit in. It is emphasized that genetic algorithms are competitive only for a particular class of problems – complex, large-scale optimization problems, as a rule. Furthermore, a problem taxonomy of interest to chemometrics, and beyond, is proposed, spanning: NUM (numerical parameter estimation) problems, SEQ (sequencing) problems, and SUB (subset selection) problems. Finally, the schema theorem is treated in a simplified way – thereby emphasizing the importance of the so-called building block principle and the crucial role of the recombination operator therein.

Chapter 2 is the second part of a tutorial on genetic algorithms. Important practicalities concerning representation, configuration, and hybridization are treated for the NUM-, SEQ-, and SUB problem types.

As for representation, regular- and Gray binary encoding for NUM problems are treated; permutation encoding for SEQ problems is treated; binary- and direct subset encoding for SUB problems are treated, laying emphasis on the latter kind.

As for configuration, it is underscored that good search performance depends critically on the extent to which the building block principle is complied with, and a balance between exploration and exploitation is attained. A taxonomy of exploration operators – recombination and mutation – according to abovementioned problem types is given; here, preservational properties of recombination operators are discussed in detail. Exploitation operators, too, are treated; it is explained why their optimal choice hardly depends on the nature of the target problem.

As for hybridization, several workable lines of approach are presented; these elucidate how genetic algorithms can be extended to enhance overall performance.

Part 2. Development

Chapter 3 is the first part of a software-oriented treatise on genetic algorithms. After a general introduction, the main focus is on requirements and viable strategies for the development of genetic algorithm applications. This discussion should serve the novice practitioner who is confronted with discouraging hurdles caused by the versatility and related poor configurability of genetic algorithms. The strategies are based on a proposed software taxonomy, spanning (in order of increasing prerequisite expertise): education software, application software, and development software. The properties and likely further evolution of these types of software are briefly passed in review.

Chapter 4 is the second part of a software-oriented treatise on genetic algorithms. It describes GATES – Genetic Algorithm Toolbox for Evolutionary Search – a "home"-made library of domain-independent routines aimed at supporting the development of genetic algorithm applications; all applications described in this thesis were developed using GATES. The routines comprising GATES are organized in logical modules. Each module features different levels of abstraction (hierarchical structure), so that routines in outer layers hide procedural details of routines in inner layers. The outmost layer comprises special routines, called prototypes, or genetic shells. Basically, a prototype is an almost complete genetic algorithm, missing only the objective function and possibly some other domain-dependent routines. GATES incorporates three prototypes, for NUM-, SEQ-, and SUB problems, respectively. The prototype for NUM problems is treated in most detail. Additionally, an auxiliary utility for configuration, based on explorative walks in the search space, is outlined.

Chapter 5 introduces a general-purpose subset recombination operator for so-called fixed-size SUB problems: subset selection problems wherein

the size of the subset is regarded constant. The development of this operator is motivated by the coincidence that SUB problems have up to now apparently enjoyed only little attention in the science of genetic algorithms, yet problems of this kind traditionally enjoy considerable interest within analytical chemistry.

The general-purpose subset recombination operator can be used in two basic preservational modes: "regular" and "sequenced". In regular mode, the placement of items in the subset is unimportant. In sequenced mode, sub-modes are available for optimal preservation of item order and -position, respectively.

Part 3. Application

Each chapter in this part treats an application of a genetic algorithm to a complex, large-scale optimization problem within analytical chemistry. Thereby, considerable attention is devoted to configuration; in particular, the choice of the recombination operator is elucidated wherever this is regarded illustrative. By far most attention is spent to the objective (fitness) function, for it defines the application domain. In all applications, this function may be viewed as basically a similarity criterion that quantifies how close a given candidate solution reaches some predefined goal.

In general, the results of these studies are promising, thus proving that genetic algorithms can be a competitive asset to chemometrics, especially when it comes to complex, large-scale optimization. However, each different application demands its own, time-consuming configuration procedure, since up to now no standard methodology appears to be available to that end. For this reason, further research in which high priority is allotted to the improvement of the configuration methodology, seems sensible.

Chapter 6 presents DENISE - a genetic algorithm for conformational analysis of DNA, using 2D-NOE (NMR) spectra and/or known H–H distance ranges as experimental constraints. The search is performed in torsion angle space, i.e. the problem is of the NUM type. DENISE was applied to a simulated 2D-NOE spectrum, calculated from a known DNA conformation to allow verification; some noise was added to the spectrum in order to approximate experimental conditions. DENISE found the target conformation within experimental error. The chapter concludes with proposing local search as a potential strategy for further improvement of the result (post-hybridization).

Chapter 7 presents, in continuation of Chapter 6, another study involving DENISE – in this case for conformational analysis of a dinucleotide photodimer, using experimental (real) 2D-NOE spectra. Again, it is shown that DENISE samples the search space in an efficient and robust way. In addition, it is shown that the result can be further improved by subsequent molecular mechanics calculations, i.e. by local search based on energy minimization as a post-hybridization strategy.

Chapter 8 presents a genetic algorithm for searching an optimal working point on a multivariate, multi-response calibration graph of dry reagent strips for quantitative clinical analysis. The search space of this NUM problem is spanned by chromatographic variables such as the mobile phase composition, pH value, etc. The responses are the intensities of a number of emission lines in an atomic spectrogram of the mixture to be analyzed. The optimal working point is the set of variable values for which the responses are simultaneously maximal according to an Euclidean distance criterion applied in the response space. The landscape of this scalar response, being an arithmetic composition of the separate intensities, can be considerably complex. Another difficulty in solving the problem resides in the fact that the response relates non-bijectively to the variables, i.e. different sets of variable values may give rise to the same response. Under these circumstances, search methods (i.e. inductive or indirect strategies) are normally a better choice than analytical methods (i.e. deductive or direct strategies). This premise is corroborated in a comparative study involving the genetic algorithm as a search method and nonlinear PLS regression as an analytical method, both applied to abovementioned problem.

Chapter 9 presents a genetic algorithm for peak deconvolution by curve fitting to interpret spectra – a NUM problem. The spectra used in this study are X-ray equator diffractometer scans of poly(ethylene)naphthalate yarns; these spectra feature strongly overlapping peaks. The search task consists in finding values for parameters that describe a number of peaks assumed to be present in the input spectrum. Each peak is described

by four parameters according to the Pearson VII model for peak forms. This model encompasses, inter alia, pure Gauss forms, pure Lorentz forms, and mixed forms thereof.

The rationale for a evolutionary approach to curve fitting resides in the fact that traditional search techniques applied to the problem have a notorious record of unreliability; in particular, the results turn out to depend critically on the required initial estimate of the solution. Genetic algorithms, on the other hand, have a reputation of robustness: they still perform well with less accurate initial estimates of the solution.

In this study, the initial estimate for the genetic algorithm is provided by a simple artificial neural network trained by using comparatively simple simulated spectra as input. The resulting hybrid searching system, thus composed of two different forms of natural computation, is used routinely at a large chemical company in the Netherlands since recently.

Chapter 10 describes CFIT – the executable genetic algorithm (program) used for the study outlined in Chapter 9. In addition to the hybridization strategy described in the latter chapter, some alternative hybridization approaches aimed at reduction of the overall running time, are mentioned.

Chapter 11 presents a genetic algorithm for the sequential assignment of 2D-NOE (NMR) spectra of proteins; this sequenced, fixed-size SUB problem provides a good test-case for the general-purpose subset recombination operator discussed in Chapter 5. The problem is heavily underconstrained since in most cases more patterns are available than amino acid positions, and uncertainties may exist in the preliminary assignments. It is shown that even in the presence of a considerable amount of errors in the input data, acceptable solutions can be found.

Chapter 12 presents a pilot study into the background, feasibility, and competitiveness of GCA – a genetic algorithm for k-medoid partitional clustering of large experimental data sets: a regular, fixed-size SUB problem. In this study, performance was ranked with respect to CLARA – an alternative, recently introduced, reportedly promising approach to k-medoid clustering of large datasets. It is shown that GCA produces considerably better results for data sets that contain many clusters.

Chapter 13 presents a comparative study involving a genetic algorithm, simulated annealing, and stepwise elimination, as methods for wavelength selection in multi-component analysis: an unknown-size SUB problem. The wavelength selection criteria used are the selectivity and accuracy after Lorber, and the minimal mean squared error after Sasaki. The genetic algorithm generally performed best. Stepwise elimination performed surprisingly good despite its local search heuristic. Simulated annealing performed worst, which is remarkable in view of the fact that this method is widely praised in the literature for properties similar to those of genetic algorithms, e.g. a probabilistic, non-local search heuristic.

Samenvatting

Het volgende is een samenvatting van de hoofdstukken 1 t/m 13.

Deel 1. Verwerving

Hoofdstuk 1 is het eerste deel van een onderricht over genetische algoritmen. Aan de orde komen de basisprincipes van genetische algoritmen, alsmede hun belangrijkste eigenschappen en het breder methodologisch verband waar zij in passen. Benadrukt wordt dat genetische algoritmen alleen concurrerend zijn voor een bepaalde klasse van problemen, t.w. complexe problemen met een grote zoekruimte. Verder wordt een probleemtaxonomie geïntroduceerd welke van belang wordt geacht voor de chemometrie (alsmede daarbuiten). Deze omvat NUM (numerieke parameterschatting) problemen, SEQ (rangschikking) problemen, en SUB (subset selectie) problemen. Tenslotte wordt het schema-theorema op vereenvoudigde wijze behandeld, waarbij nadruk wordt gelegd op het belang van het zogeheten bouwsteen-principe en de cruciale rol welke de recombinatie-operator daarin speelt.

Hoofdstuk 2 is het tweede deel van een onderricht over genetische algoritmen. Aan de orde komen een aantal belangrijke praktische aspecten inzake representatie, configuratie en hybridisatie voor de NUM-, SEQ- en SUB probleemtypen.

In de discussie over representatie voor NUM problemen komen reguliere- en Gray binaire codering aan de orde; voor SEQ problemen wordt permutatie-codering behandeld; en voor SUB problemen wordt licht geworpen op binaire- en directe subset codering, met nadruk op laatstgenoemde soort.

In de discussie over configuratie wordt onderstreept dat het bereiken van goede zoekprestatie kritisch afhangt van de mate waarin aan het bouwsteen-principe voldaan wordt en evenwicht tussen exploratie en exploitatie bereikt is. Voor de exploratie-operatoren – recombinatie and mutatie – wordt een op bovengenoemde probleemtypes gebaseerde taxonomie gegeven; daarbij worden behoudende eigenschappen van recombinatie-operatoren uitvoerig besproken. Tevens worden exploitatie-operatoren behandeld; uitgelegd wordt waarom de optimale keuze van deze operatoren nagenoeg niet afhangt van de aard van het doelprobleem.

Voor wat betreft hybridisatie worden diverse praktisch uitvoerbare strategieën in grote lijnen gepresenteerd; deze belichten hoe genetische algoritmen kunnen worden uitgebreid teneinde de algehele zoekprestatie te verbeteren.

Deel 2. Ontwikkeling

Hoofdstuk 3 is het eerste deel van een software-georiënteerde verhandeling over genetische algoritmen. Na een algemene inleiding is de aandacht gericht op vereisten en kansrijke strategieën voor de ontwikkeling van toepassingen van genetische algoritmen. Deze discussie dient als leidraad voor de beginnend beoefenaar die geconfronteerd wordt met ontmoedigende hindernissen veroorzaakt door de veelzijdigheid en daarmee verband houdende povere configureerbaarheid van genetische algoritmen. De strategieën zijn gebaseerd op een daartoe geïntroduceerde software-taxonomie, omvattende (in volgorde van toenemend vereiste praktische ervaring): software voor training, -toepassing en -ontwikkeling. De eigenschappen en waarschijnlijke verdere evolutie van deze software-typen passeren kort de revue.

Hoofdstuk 4 is het tweede deel van een software-georiënteerde verhandeling over genetische algoritmen. Het beschrijft GATES – Genetic Algorithm Toolbox for Evolutionary Search – een in-"huis" ontwikkelde bibliotheek bestaande uit domein-onafhankelijke routines ter ondersteuning van de ontwikkeling van toepassingen van genetische algoritmen; alle in dit proefschrift beschreven toepassingen zijn ontwikkeld vanuit GATES. De routines in GATES zijn georganiseerd in logische modules. Iedere module wordt gekenmerkt door verschillende abstractie-niveaus, zodat routines in buitenste lagen de procedurele details van routines in binnenste lagen afschermen. De buitenste laag bestaat uit speciale routines, welke ook wel prototypes of genetische schillen worden genoemd. In essentie is een prototype een bijna-

compleet genetisch algoritme waarin alleen nog maar de doelfunctie en mogelijk andere domeinafhankelijke routines dienen te worden ingevuld. GATES bevat drie prototypes, t.w. voor NUM-, SEQ- en SUB problemen. Het prototype voor NUM problemen wordt het meest uitvoerig behandeld. Verder wordt een utiliteit ter ondersteuning van configuratie behandeld; deze is gebaseerd op exploratieve wandelingen in de zoekruimte.

Hoofdstuk 5 introduceert een universele subset recombinatie-operator voor zogenaamde vastegrootte SUB problemen: subset selectie-problemen waarin de grootte van de subset constant wordt verondersteld. De motivatie voor de ontwikkeling van deze operator ligt in de toevallige samenloop van omstandigheden dat SUB problemen tot op heden kennelijk nog maar betrekkelijk weinig aandacht hebben genoten in het vakgebied der genetische algoritmen, terwijl in de analytische chemie traditioneel juist wel veel belangstelling voor dit soort problemen bestaat.

De universele subset recombinatie-operator kan op twee elementaire manieren voor behoud van eigenschappen worden gebruikt: "regulier" en "gerangschikt". In het reguliere geval wordt geen rekening gehouden met de plaatsing van de elementen in de subset. In het gerangschikte geval daarentegen wel; er kan dan worden gekozen voor behoud van ofwel de volgorde ofwel de positie van elementen in de subset.

Deel 3. Toepassing

Ieder hoofdstuk in dit deel behandelt een toepassing van een genetisch algoritme op een complex analytisch-chemisch probleem met een grote zoekruimte. Hierbij wordt veel aandacht geschonken aan configuratie; in het bijzonder wordt de keuze van de recombinatie-operator overal toegelicht waar dit verduidelijkend wordt geacht. Verreweg de meeste aandacht wordt besteed aan de doel(fitness-)functie, aangezien deze immers het toepassingsdomein definieert. Deze functie mag in alle toepassingen in wezen worden beschouwd als een gelijkeniskriterium welke kwantificeert in hoeverre een gegeven kandidaatoplossing een vooraf gedefinieerd doel bereikt.

Over het algemeen zijn de resultaten van deze studies veelbelovend, zodat aangetoond is dat genetische algoritmen een concurrerende aanwinst zijn voor de chemometrie, met name wanneer het gaat om complexe optimalisatie op grote schaal. Evenwel vereist iedere verschillende toepassing zijn eigen, tijdrovende configuratieprocedure, aangezien er tot op heden geen standaard-methodologie voor dergelijke doeleinden beschikbaar blijkt te zijn. Derhalve lijkt vervolg-onderzoek waarin hoge prioriteit wordt geschonken aan de verbetering van de configuratiemethodologie zinvol.

Hoofdstuk 6 presenteert DENISE – een genetisch algoritme voor conformatie-analyse van DNA, waarbij 2D-NOE (NMR) spectra en/of bekende H–H afstandsintervallen worden gebruikt als experimentele gegevens. Het zoeken vindt plaats in de door torsiehoeken opgespannen zoekruimte, d.w.z. het betreft een NUM probleem. DENISE werd toegepast op een nagebootst 2D-NOE spectrum, berekend vanuit een bekende DNA conformatie, teneinde verificatie mogelijk te maken; enige ruis werd toegevoegd aan het spectrum teneinde experimentele omstandigheden te benaderen. DENISE vond de doelconformatie binnen de experimentele fout. Het hoofdstuk besluit met het opperen van lokaal zoeken als een mogelijke strategie voor verdere verbetering van het verkregen resultaat (post-hybridisatie).

Hoofdstuk 7 presenteert, in vervolg op hoofdstuk 6, een andere studie aangaande DENISE – in dit geval voor conformatie-analyse van een dinucleotide fotodimeer, waarbij experimentele (echte) 2D-NOE spectra worden gebruikt. Wederom wordt aangetoond dat DENISE de zoekruimte op efficiënte en robuuste wijze bemonstert. Tevens wordt aangetoond dat het resultaat verder kan worden verbeterd door moleculaire mechanica-berekeningen, d.w.z. door op energie-minimalisatie gebaseerd lokaal zoeken als een post-hybridisatie strategie.

Hoofdstuk 8 presenteert een genetisch algoritme voor het zoeken naar een optimaal werkpunt op een multivariate multirespons ijkgrafiek van droge reagensstrips voor kwantitatieve analyse. De zoekruimte van dit NUM probleem wordt opspannen door chromatografische variabelen zoals de compositie van de mobiele fase, de pH-waarde, etc. De responsen zijn de intensiteiten van een aantal emissielijnen in een atoomspectrogram van het te analyseren mengsel. Het optimale werkpunt is de verzameling met waarden voor de variabe-

len waarvoor de responsen tegelijkertijd maximaal zijn volgens een Euclidisch afstandskriterium toegepast in de responsruimte. Het landschap van deze scalar-respons, zijnde een rekenkundige compositie van de afzonderlijke intensiteiten, kan aanmerkelijk complex zijn. Een andere moeilijkheid bij het oplossen van dit probleem schuilt in het feit dat het verband tussen de respons en de variabelen niet-bijectief is, d.w.z. verschillende verzamelingen met waarden voor de variabelen kunnen leiden tot dezelfde respons. Onder deze omstandigheden zijn zoekmethoden (oftewel inductieve of indirecte strategieën) normaliter een betere keuze dan analytische methoden (oftewel deductieve of directe strategieën). Deze veronderstelling wordt bevestigd in een vergelijkend onderzoek aangaande het genetisch algoritme als zoekmethode en niet-lineaire PLS regressie als analytische methode, beide toegepast op bovengenoemd probleem.

Hoofdstuk 9 presenteert een genetisch algoritme voor piek-deconvolutie d.m.v. curve-passen ter interpretatie van spectra – een NUM probleem. De in deze studie gebruikte spectra zijn metingen aan poly(ethyleen)naftalaat-garens uitgevoerd op een röntgen-diffractometer; deze spectra kenmerken zich door sterk overlappende pieken. De zoektaak behelst het vinden van de waarden voor de parameters welke een aantal in het spectrum veronderstelde pieken beschrijven. Iedere piek wordt door vier parameters beschreven volgens het Pearson VII model voor piekvormen. Dit model beslaat ondermeer zuivere Gauss-vormen, zuivere Lorentz-vormen, en mengvormen daarvan.

De aanleiding voor een evolutionaire benadering tot piek-deconvolutie d.m.v. curve-passen ligt in het feit dat traditionele zoektechnieken toegepast op het probleem er over het algemeen berucht om staan onbetrouwbare resultaten op te leveren; met name blijkt dat de resultaten sterk afhangen van de vereiste beginschatting van de oplossing. Genetische algoritmen daarentegen hebben een reputatie van robuustheid: met minder nauwkeurige beginschattingen van de oplossing worden nog steeds acceptabele resultaten verkregen.

In deze studie wordt de beginschatting voor het genetisch algoritme aangeleverd door een eenvoudig kunstmatig neuraal netwerk dat getraind is op basis van betrekkelijk eenvoudige nagebootste spectra. Het resulterend hybride zoeksysteem, aldus bestaande uit twee verschillende vormen van natuurlijke intelligentie, wordt sinds kort routinematig gebruikt in een groot chemisch bedrijf in Nederland.

Hoofdstuk 10 beschrijft CFIT – het uitvoerbare genetisch algoritme (programma) gebruikt in de studie behandeld in hoofdstuk 9. In toevoeging op de in laatstsgenoemd hoofdstuk beschreven hybridisatie-strategie, worden een aantal alternatieve, eveneens op het reduceren van de totale rekentijd gerichte hybridisatie-strategieën aangestipt.

Hoofdstuk 11 presenteert een genetisch algoritme voor de sequentiële toekenning van 2D-NOE (NMR) spectra van eiwitten; dit gerangschikt vaste-grootte SUB probleem is geschikt voor het uittesten van de universele subset recombinatie-operator behandeld in hoofdstuk 5. Het probleem is sterk onderbepaald aangezien in de meeste gevallen meer patronen beschikbaar zijn dan aminozuurposities, en er onzekerheden in de voorafgaande toekenningen kunnen bestaan. Aangetoond wordt dat zelfs bij een vrij grote hoeveelheid fouten in de invoergegevens nog acceptabele oplossingen kunnen worden gevonden.

Hoofdstuk 12 presenteert een verkennend onderzoek naar de achtergronden, haalbaarheid en meerwaarde van GCA – een genetisch algoritme voor k-medoide partitioneel clusteren van grote gegevensverzamelingen: een regulier vaste-grootte SUB probleem. In deze studie wordt een vergelijking gemaakt met het recentelijk geïntroduceerde en volgens de literatuur veelbelovende CLARA – een alternative benadering voor k-medoide partitioneel clusteren van grote gegevensverzamelingen. Aangetoond wordt dat GCA beduidend betere resultaten oplevert voor gegevensverzamelingen met veel clusters.

Hoofdstuk 13 presenteert een vergelijkend onderzoek betreffende een genetisch algoritme, "nagebootst afkoelen" (simulated annealing), en "stapsgewijze verwijdering" (stepwise elimination), als methoden voor golflengte-selectie in multicomponenten-analyse: een variabele-grootte SUB probleem. Als kriteria voor golflengte-selectie worden gebruikt de selectiviteit en nauwkeurigheid volgens Lorber, en de minimale gemiddelde kwadratische fout volgens Sasaki. Het genetisch algoritme leverde over het algemeen de beste resultaten. Stapsgewijze verwijdering bleek ech-

ter verrassend goed te werken ondanks diens lokale zoekheuristiek. Nagebootst afkoelen leverde de slechtste resultaten, hetgeen opmerkelijk is gezien het feit dat deze methode in de literatuur vaak wordt geprezen voor eigenschappen gelijkend op die van genetische algoritmen, bijvoorbeeld een probabilistische, niet-lokale zoekheuristiek.

Curriculum Vitae

(Geschreven door Ann Yee)

Carlos B. Lucasius volgde het ongedeeld VWO aan het Maria Immaculata Lyceum op Curaçao, Nederlandse Antillen. In hetzelfde jaar waarin hij hier voor slaagde vertrok hij naar Nederland voor de studie chemie aan de Katholieke Universiteit te Nijmegen.

Aan genoemde universiteit volgde Carlos het kandidaatsprogramma S3 (hoofdvakken scheikunde en natuurkunde). De doctoraal fase welke hierop volgde bestond uit de hoofdrichtingen analytische chemie (Prof. G. Kateman) en vaste stof chemie (Prof. P. Bennema), uitgebreid met capita in de Katalyse (Prof. J.W.E. Coenen). In deze periode is hij enige tijd werkzaam geweest aan de voormalige olieraffinaderij Shell Curaçao NV.

Van juni 1988 tot juli 1992 was Carlos als wetenschappelijke assistent verbonden aan de vakgroep Analytische Chemie van de Katholieke Universiteit te Nijmegen waar het in dit proefschrift beschreven onderzoek werd verricht. In deze periode begeleidde hij acht doctoraalstudenten. Hij verzorgde avondcursussen in de chemometrie-onderwerpen Signaalverwerking en Patroonherkenning aan een hogere laboratoriumschool in Den Haag en in Utrecht. Hij was lid van de landelijke werkgroep Optimalisatie van de Koninklijke Nederlandse Chemie Vereniging. Hij gaf presentaties op de diverse (inter)nationale conferenties en symposia. Hij werkte vruchtbaar samen met een aantal interne- en externe onderzoeksinstituten – universiteiten en bedrijven. Het ging hierbij om verschillende toepassingsgebieden van genetische algoritmen; in een enkel geval werden de resultaten onderdeel van een octrooi.

Ook na de afronding van het promotie-onderzoek zochten externe instanties samenwerking met de vakgroep voor de toepassing van genetische algoritmen. Dit heeft mede geresulteerd in de samenwerkingsprojecten OMEGA (Optimization of Metric matrix Embedding using Genetic Algorithms), met als partner (en financierder) CIBA-GEIGY AG in Zwitserland, en RENEGADE (REsearch NEtwork on Genetic Algorithms DEvelopment, gefinancieerd door de Commissie van de Europese Gemeenschappen), met als partner BRAINWARE GMBH in Duitsland. Bij de aanvraag en het op gang brengen van deze projecten heeft Carlos een substantiële bijdrage geleverd

Na de verdediging van dit proefschrift zal Carlos o.b.v. Prof. P.D. Wentzell aan Dalhousie University te Halifax, Canada, postdoctoraal onderzoek verrichten naar *natural computation* toegepast in chemische signaalverwerking. Voor dit onderzoek werden twee Canadese stipendia aangevraagd: een *Killam Postdoctoral Fellowship* (KILLAM TRUSTS aan Dalhousie University te Halifax) en een *Canadian International Fellowship* (NATURAL SCIENCES AND ENGINEERING RESEARCH COUNCIL te Ottawa). Beide stipendia werden toegekend, voornamelijk op basis van de resultaten voortvloeiend uit het in dit proefschrift beschreven onderzoek Carlos accepteerde uiteindelijk laatstgenoemd stipendium.